TAMPERE UNIVERSITY OF TECHNOLOGY

JiaJing Tan

GAME PHYSICS ON REAL-TIME SIMULATION OF MOBILE WORK MACHINE

Master's thesis

Examiner: Professor Doctor Asko Ellman
Examiner and topic approved by the Faculty Council of the Faculty of Mechanics and Design, Product Engineering on Feb,2012

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Degree program in Machine Automation
STUDENT, JIAJING TAN (217225), GAME PHYSICS ON REAL-TIME SIMULATION OF MOBILE WORK MACHINE
Master of Science Thesis, 83 pages with Appendix 7 pages
April 24, 2012
Major subject: Mechatronics
Examiner: Professor Dr.Tech Asko Ellman
Keywords: Blender, Game Physics, Collision boundaries, rigid body dynamics, Lagrangian Formula, Forward Kinematics, Virtual Reality.

Game physics has been around since the first game were created around thirty years ago, it was first developed from sparks fireworks, ballistics of bullets to a physical simulation of human skeleton in most recent years. With its rapid development, game physics expanded its usage from virtual game into various tasks in numerous areas. The thesis tests usage and reliability of the game physics into mobile machine simulation. It compares machine simulation in Matlab, as traditional programming approach, with the result in Blender game engine, a 3D application integrated with game physics. The process of developing dynamic behaviors in game physics is separated into several main steps: importing of ready-made model, setting up of physical properties, creation of interactive logic controls and finally position calculation of crane via Python scripting. Different results regarding game physics are collected and compared alongside with two methodologies and thus final evaluation of Game physics and its implementation in mobile machine area are given in the end.

The quality of simulation as an example provides a high degree of realism and interaction between users and computers than traditional approaches. During the process, reader could understand the totally different modeling and simulation styles existing in two methodologies, Matlab and Blender game engine. The thesis offers readers one idea about game physics and how physics could be integrated into and thus being able to distinguish the advantage and disadvantage of game physics and get an idea on the future state of virtual reality technology in general.

Acknowledgements

The thesis is carried out under the department of Mechanics and Design in Tampere University of Technology.

The author would like to express her gratitude to all those who confirm the permission and thus make it possible to complete the thesis work at Tampere University of Technology specially Mechanics and Design department.

Most notably, I would like to convey my grateful thanks to the thesis supervisor and examiner Professor Asko Ellman for providing numerous advisory supports and professional advices of any kind on thesis and his great kindness on permission of carrying out the thesis abroad. I also want to thank research assistant Mr. Ville Lepokorpi and Mr. Joonatan Kuosa both at Tampere University of Technology for giving me numerous advisable helps regarding Lagrangian equations in Matlab simulation, ode solver and introduction lecture about Blender physics. Furthermore, I am deeply thankful to Mr. Diogo Nuno Mendes Tavares at Tampere University of Technology for his outstanding and warm encouragement from the very beginning of this thesis work as well as for his intensive and valuable discussion about thesis scientific writing skills. I would also like to thank HIAB for providing their crane model with reliable dimensions. Last but not least, I would like to give my special thanks to all my beloved friends in Finland and China who have been encouraging and inspiring me for the duration of thesis preparation and composing period.

JiaJing Tan
Shanghai, China

Structure of the thesis

The thesis consists of 5 major parts and thus is organized as follows:

The first chapter provides an overview of the objective, purpose and motivation of thesis briefly.

The second chapter introduces the entire development of Mathematic model of crane via Matlab programming. An efficient approach is developed to a general solution of hydraulic crane in terms of forward kinematics and dynamics equation.

Chapter 3 contains the background introduction of Blender, game engine and game physics as well as the development of hydraulics crane which can be divided into 3 major steps.

Chapter 4 and 5 give a detail discussion about the pros and cons of Blender game physics as representation of game physics technology compared with Matlab, traditional modeling approaches in several aspects.

Chapter 6, which is the last chapter, gives the final conclusion for whole thesis and predicts the future trends of game physics technology will be.

Lists of symbols, terms and figures

Symbols and terms

$\boldsymbol{c_i}$	Center of objects i
$\boldsymbol{cg_i}$	Center of gravity of 1st,2nd and 3rd cylinder
$\boldsymbol{cg_{ti}}$	Center of gravity of 1st,2nd and 3rd torque link
$\boldsymbol{c_{yi}}$	Position of each cylinder in their coordinates
$\boldsymbol{d_i}$	Difference between Blender game engine and Matlab calculation
$\mathbf{f}_i$	Resulting force at each cylinder rod
$\mathbf{g}$	Gravity acceleration vector
$\mathbf{I}$	Inertia tensor of an object
$\mathbf{J}$	Jacobian Matrix
$\boldsymbol{k}$	Kinetic energy
$\boldsymbol{l_i}$	Length of each link i
$\mathbf{m}$	Mass of an object
$\mathbf{M}$	Mass matrix
${}^{i-1}_{i}\boldsymbol{P}$	Position vector of frame {i} with respect to frame {i-1}
$\mathbf{q}$	Vector of generalized coordinates
$\dot{\mathbf{q}}$	Derivation of generalized coordinates
$\boldsymbol{r_i}$	Radisu of cylinder tube
$\boldsymbol{r_{rod}}$	Radius of cylinder rod
${}^{i-1}_{i}\boldsymbol{R}$	Rotation matrix from frame {i} to {frame i-1}
$\boldsymbol{t_i}$	Joint center of torque link i
${}^{i-1}_{i}\boldsymbol{T}$	Transformation matrix from frame {i} to {frame i-1}
$\mathbf{u}$	Potential energy
${}^{i}\boldsymbol{w_i}$	Angular velocity with respect to its own frame {i}
$\{\boldsymbol{X_i}, \boldsymbol{Y_i}\}$	Frame originate at point I
$\boldsymbol{x_{pi}}$	Piston extension that cylinders i have passed
$\boldsymbol{\alpha_{xp}}$	The angle corresponding to cylinder extension
$\boldsymbol{\theta_1}$	1st joint variable
$\boldsymbol{\theta_2}$	2nd joint variable
$\boldsymbol{\tau_i}$	Torque of link i

Table of Contents

1. Introduction

1.1 Objective of the thesis

The object of thesis is a forestry hydraulic crane, type HIAB Log lift 105 S. It is responsive and accurate control, combined with sufficient reach, guarantee smooth loading. This foldable and light weight crane makes full use of both cargo space as well as carrying capacity. (Hiab Loglift 105)

Figure 1: A typical kind of forestry crane in forest-based industry (Forestry cranes)

The machine consists of a frame based design and equipped with very powerful joints and links actuated by hydraulic cylinders and pumps. The entire rotation is achieved by two rotational joints between the base, first link and second link of the crane model. The last joint could be attached to a telescope with a prismatic joint or a gripper that is rotatable to reach the whole working area often.

This kind of cranes are equipped with lowlighted-structure and hydraulic actuators and thus they have a high ground pressure which allows climbing over rocks, stumps and travelling through deep snow or wet lands as well. Hence, forestry cranes have excellent stability which makes them be able to use full reachable and lifting capacities. (Heinze, 2007) Therefore, this typical kind of forestry mobile machines are additionally utilized in operations for falling, climbing and cutting trees as well as transforming log from the stump to a roadside landing in the forest industry.

1.2 Goal of the thesis

The mobile machine has applied to its usage into various kinds of different fields, for

example, small forestry crane for transporting material for paper manufacture or farm, and heavy-duty construction crane for construction projects. Regardless various sizes of different mobile machines, it is still needed to build better and efficient techniques to ensure that safety factors and efficient working loads are met during daily operation.

With the help of rapid development and advancement in computer graphics and hardware, virtual reality technology makes it possible to provide a reliable environment for testing, training and simulation before start. As the virtual reality technology has developed throughout years, many researchers have created methods to generate animations for training operations and processes (Wei Guoqian; Zhang Zhenyan; Dong Haoming, 2009). However, to some extent, the simulation generated by previous developing game render had logic errors due to ignorant of the physical laws in the game. In order to have a reliable and realistic simulation, complicated behaviors of mobile machine and general environment that exist in real life should be taken into consideration, such like the effect of gravitational forces, friction effect between objects and physical bounds of objects. Therefore, virtual game should engage physics effect such as rigid body, soft body dynamics into simulation. By simulating realistic motions of objects with physics specification, it can help to evaluate whether designed performances of objects are applicable or not in advance.

A simulation should be created with specific physics information on environment and objects. These will include object geometry, time factors, dynamic motions which are commonly ignored in conceptual animation or animated simulation. All consideration will be helpful to figure out the potential challenges in the existing condition, such as inefficient operation of machines, spatial conflicts on the working site.

As a conclusion, game physics technology provides a fast way to assess the performance of any object correctly in virtual environment. Thus, making virtual simulation of any object based on game physics is necessary in the development of any product before its start of production.

1.3 Previous works and researches on Game Physics

Game Physics was first created in a games which were written around thirty years ago. It was first seen in the way how particles move: sparks fireworks, the ballistics of bullets, smoke, and explosion. Next came the car physics, with ever increasing sophistication of tire, suspension and engine models. As processing power become available, it is possible to have crates that could be moved around or stacked, walls that could be destroyed and crumble into constituent blocks. Thus, rigid body physics, which rapidly expanded to include softer objects like clothes, flags and rope. Most recently, there is a rise of the ragdoll: a physical simulation of the human skeleton that allows more realistic trips, falls and death throes into game. (Millington, 2007)

There have been many advanced researches to improve reliability of game physics and thus many physics engines have been developed to increase strong capacities in a

speedy, stable and robust application in recent years. Hence, researchers have started utilizing the advantages of game physics in simulation in various fields, such as developing in robotics, mobile machine and machine for construction practice. Huang J.Y and Gau C.Y, (2003) developed a mobile machine crane simulator which took collision detection of crane into consideration. The reference report from (Chi H L, Hung W.H,Kang S.C, 2007) has used the physics engines to model the physics of a construction crane and simulate its activities. These researches are an example of numerous investigation that have made been before, which demonstrates the feasibility and potential of developing simulation based on game physics.

Although physics engine facilitates developers to build models and simulate machine operation, there are still a lot of strives to put in physics modeling and programming in order to integrate with other graphics. Researchers start considering developing own specific simulators and simulation and thus, it could be run in game engine as an additional plug-in.

As a conclusion, developing simulations via game technology can provide inexpensive state-of-art 3D virtual worlds in a short period. Users could make simulation via game engine without extensive programming. (Trenholme D and Smith S.P., 2008)

1.4 Challenges in existing game physics

The challenges in existing game physics are mainly virtual environment and physics effect. The report by Lewis and Jacobson (2002) indicates that high quality of virtual reality and physical simulation has been a barrier of entry for most research developers in virtual reality. Developing virtual environments from the ground up is prohibitively costly, complicated and time consuming. The development of game physics needs a combination of technology, knowledge and skills. In order to implement high levels of physics simulation, it is necessary to address important requirements: high quality graphics, realistic physics effects and a highly efficient system. (Simith S.P,Duke D.J, 2000)

Physics effect such as the effect of gravity, collision response and other physics effect are simulated with mathematical computations and algorithms derived from physics law. Go back to game render time, game developers have to write own methods or algorithms and combine them into the rendering world. In particular, the developers need to combine physics knowledge with mathematical skills and concentrate programming efforts to complete the system. Furthermore, real-time interaction is also required to deliver a greater range of application and usage of simulation. In order to support a real-time interactive simulation, the efficiency of the system must be a matter of concern. (J R Juang,W H Hung,S C Kang, 2010)

Luckily, game physics has revolute to reach better performance with computer graphics and interactive user interface. Therefore, it is proposed to have a real-time simulation based on game physics in Blender game engine without spending time on

low-level detailed programming.

1.5 Existing methodologies in the thesis

During this work, several computer-aided tools which have been used are described in the following section. The main Blender virtual crane model has been created in advance. Dimension and other parameters of the crane can be read explicitly with Catia drawing. The CatiaV5 R20 environment generates a lot of screenshots and used in the project for correlation of crane linkage and torque transmission.

Afterwards, the whole establishment of Mathematic model will be modeled in Matlab R2010a that allows solving the derived differential ordinary equation with a form related to the state space model. The existing methods for generating physics model in Matlab can be divided into two categories, forward kinematics to define relations and derivation of dynamics equation via Jacobian matrix.

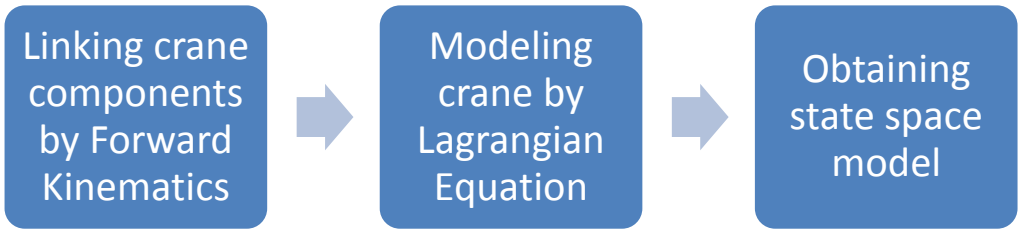


Figure 2: Flow process of development of crane in Matlab scripts.

Finally, crane experiment will be implemented using Blender R2.6 with an up-to-date plug-in Bullet Physics. It is a typical game physics partitions the physics simulation into two phases, collision detection and multi-bodies dynamics. The collision detection is a process that determines contact behaviors between joints, while multi-bodies dynamics calculate the dynamical motion of objects which are connected with each other by joints or simply exists independently. (Erleben K., Sporring J.,Henriksen K. and Dohlmaan H., 2005) The methodology developed and how process interacted with game engine in the Blender is shown in the following figure.

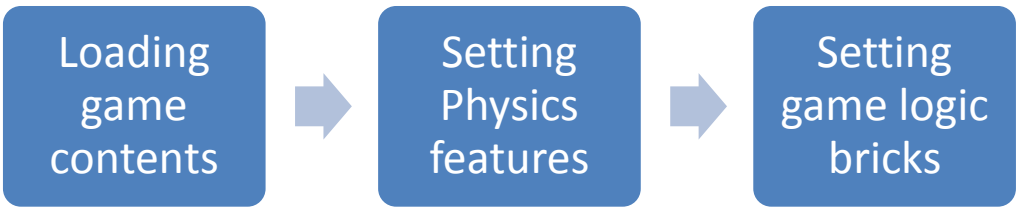


Figure 3: Flow process of development of crane in Blender.

During the entire physics modeling, there are two major challenges. One of them is constructing models and physics environment abstractly via programming in Matlab, the other is the dynamics tests in Blender game engine as its Game physics is intended

for simple simulation and still under development for advance usage out of game field. Therefore, certain tolerances of accuracy on Game Physics are allowed. Finally, concrete measures, conclusions and suggestions on Blender Game Physics will be made according to results collected from both methodologies.

2. Crane simulation in Matlab

2.1 Definition of crane components

Design and construction of a typical forestry crane is an important part of kinematics and dynamic modeling in Matlab. The manipulator includes the truck, cabin and boom of crane and gripper. However, as to simplify the model, only booms and revolute joints of crane in the planar space will be taken into modeling.

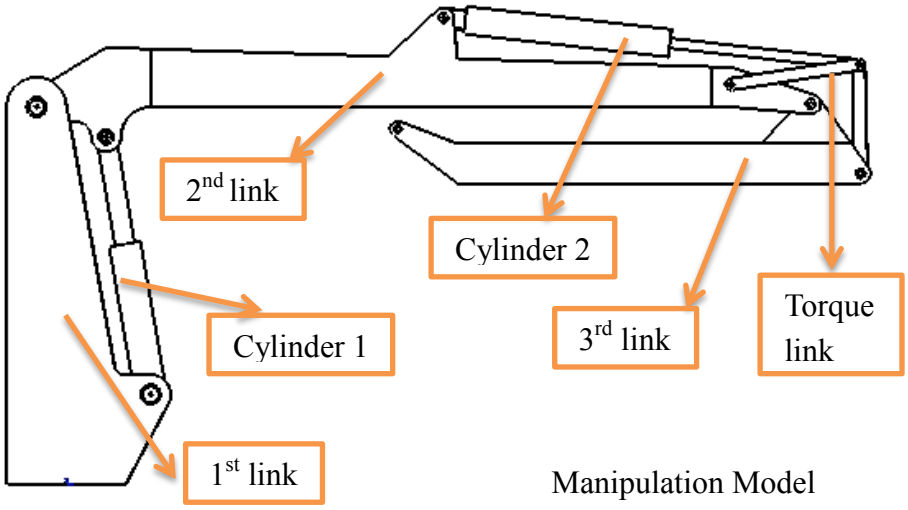


Figure 3: The architecture of the forestry crane log lift 105S model.

The 1st, 2nd, 3rd links which are regard as rigid bodies' booms and actuating cylinders are briefly introduced in the following section 2.11 and 2.12 below.

2.1.1 Rigid booms of crane

The crane consists of 2 mechanical joints motivated by two hydraulic cylinders and one mechanism of torque links which transfers the external torque from second cylinder to the third link instead of having another cylinder on the 3rd link to control its movement. The crane can move in planar space because the first link is fixed to the floor and thus it is not capable of moving. Generally, all of the crane links are modeled as rigid bodies

with spring forces simulated as boundaries.

2.1.2 Actuating cylinders

There are two cylinders attached to the crane. One situates at the first link to control the angular movement of second link. Another cylinder locates at the second link to control the movement of third link by transmission from torque links. The cylinders are considered as a part of the crane boom; therefore, the masses are calculated into the total mass of booms. In this research, cylinders are used to provide external forces by changing pressures in outlets and inlets of chambers to extent or retract strokes as well as setting restriction to workspace of crane.

2.2 Kinematic of the crane

The purpose of using forward kinematics in this research is to describe the motions of the rigid part of the crane – the manipulation model presented in the figure. By describing all spatial information between rigid bodies and cylinders in respect to origins, a controllable model in a virtual environment can be developed.

2.2.1 Schematics of the crane

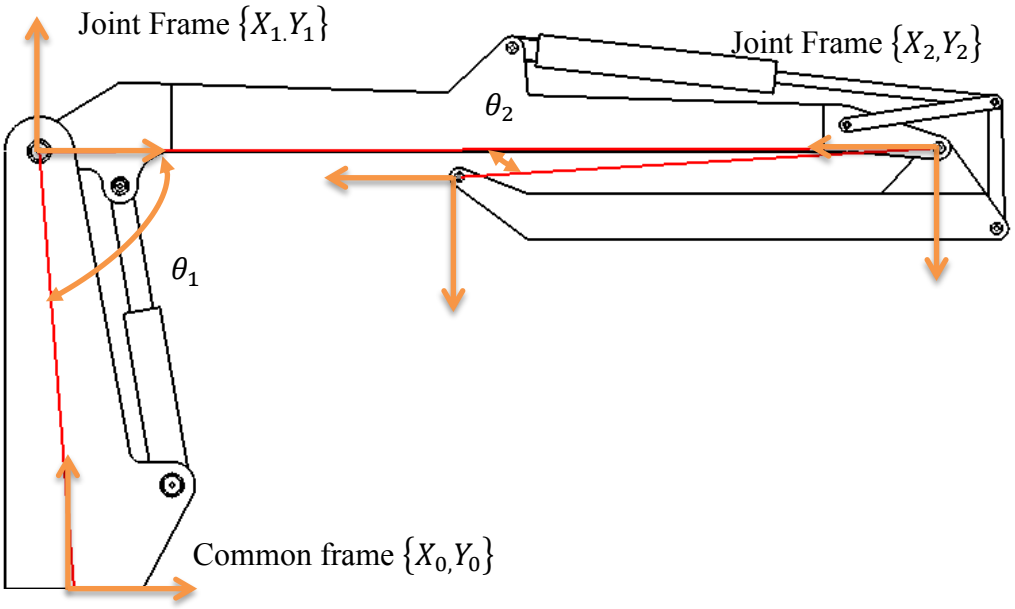


Figure 4: general definition of joint frames and joint angles of crane

The spatial information between those rigid bodies can be presented by a 4×4 matrix which describes the position and orientation of specific connection. Through the definition of rigid parts, the manipulation model of crane can be transferred into the schematics of a manipulator. Matrix of frame joints identifies each joint and describes the relative motion between neighboring links. For example, the joint between 1st link and 2nd link, or the 2nd link and torque links, torque links and 3rd link are hinges and

have finite rotation along a plane. As shown in the figure 3, the revolute joints $\boldsymbol{\theta_1}$,$\boldsymbol{\theta_2}$ denotes the working angles of crane along X-Y plane, locates at the first joint and second joint. Positions can be represented in Cartesian coordinates in terms of these angles. The base frame $\{\mathbf{x_0}, \boldsymbol{y_0}\}$, 1st joint frame $\{\mathbf{x_1}, \boldsymbol{y_1}\}$, 2nd joint frame $\{\mathbf{x_2}, \boldsymbol{y_2}\}$, 3rd crane tip $\{\mathbf{x_3}, \boldsymbol{y_3}\}$ are the positions of joints and links.

It is essential for users to identify the joint variables that establish the total frame of mechanical system. The number of variables is simply determined by system's degree of freedom. Since the crane contains 2 rotatory joints, the vector contains joint variables is regarded as,

$$\mathrm{q} = \begin{vmatrix} \theta_1 \\ \theta_2 \end{vmatrix}$$

2.2.2 Kinematics of links

2.2.2.1 Transformation matrix of joints

After constructing the schematics of the manipulator, the next step is to establish the relationship between neighboring rigid links and the transformation matrix to present the locations and orientations of booms and joints of crane. The forward kinematic can be derived by Denavit-Hartenberg (abbreviated as DH) (J.Cragic, 2005) transformation between two subsequent coordinate system. DH notation can be used to describe all kinds of manipulators by following a general procedure. In other words, it defines a coordinate system attached to each joint that are display the displacement relative to their neighbors in a world frame. Following the rules of DH procedure, four parameters $\theta_{i-1}, \mathrm{d}_{i-1}, \mathrm{a}_i, \alpha_i$ are used to describe the relationship between two coordination systems in a general form. (J.Cragic, 2005) By defining the four parameters, the transformation matrix a joint related to the previous link can be shown as,

$$ {}^{i-1}_{i}\mathrm{T} = \begin{bmatrix} {}^{i-1}_{i}R & {}^{i-1}_{i}P \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & a_{i-1} \\ \sin(\theta_i)\cos(\alpha_{i-1}) & \cos(\theta_i)cos(\alpha_{i-1}) & \sin(\alpha_{i-1}) & -\sin(\alpha_{i-1})d_i \\ \sin(\theta_i)\sin(\alpha_{i-1}) & \cos(\theta_i)\sin(\alpha_{i-1}) & \cos(\alpha_{i-1}) & \cos(\alpha_{i-1})d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

The transformation parameters can be got by translating two directions a_{i-1} and d_i and rotating α_{i-1} and θ_i along a_{i-1} and axis i-1.The DH parameters $\theta, \mathrm{d}, \mathrm{a}, \alpha$ that have defined connected to the previous joint in the procedure can be summarized into table below,

Table1: Definition of tradition DH parameters according to the (J.Cragic, 2005)

DH	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	0	l_2	0	$-\theta_2 + \pi$

In our case, the transformation matrix for the crane model is given by filling in the numbers with table 1. In this case, the transformation can be simplified to obtain as an elementary rotation of a reference frame about z axis ${}^{i-1}_{i}R_z$ and position with respect to its previous coordinate system ${}^{i-1}_{i}P$. According to figure 2, the rotation matrix from 1st joint to its origin,

$$ {}^{0}_{1}R = \begin{vmatrix} c\theta_1 & s\theta_1 & 0 \\ 0 & 0 & -1 \\ s\theta_1 & c\theta_1 & 0 \end{vmatrix} \tag{2} $$

While, the rotation matrix from 2nd joint to 1st joint,

$$ {}^{1}_{2}R = \begin{vmatrix} -c\theta_2 & s\theta_2 & 0 \\ 0 & 0 & -1 \\ -s\theta_2 & -c\theta_2 & 0 \end{vmatrix} \tag{3} $$

Where function cosine and sine satisfy the equation that c(x) = cos(x), s(x) = sin(x) and angle θ_{12} satisfies the equation that $\theta_{12} = \theta_1 + \theta_2$ respectively.

By modeling the crane in a planar frame, the rotation of the first link around z axis is out of concern. The crane tip follows the movement of the third link; therefore, the rotation of the crane tip from the second joint can be expressed through a unit matrix as shown. The rotation matrix from crane tip to 2rd joint,

$$ {}^{2}_{tip}R = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} \tag{4} $$

Eventually, after multiplying the rotation and transformation matrix, the entire 2nd and tip rotation matrix in terms of world frame have been defined:

$$ {}^{0}_{2}R = {}^{0}_{tip}R = \begin{vmatrix} -c\theta_{12} & s\theta_{12} & 0 \\ 0 & 0 & -1 \\ -s\theta_{12} & -c\theta_{12} & 0 \end{vmatrix} \tag{5} $$

2.2.2.2 Position vector of links

If crane moves forward and the location is changed with it. The general form of position at links 'center of gravity can be derived similarly as in the previous section. For example, as the center of gravity of link 1 I_{cg1} locates at $P_{1c1} = \begin{vmatrix} x_{c1} \\ y_{c1} \\ 0 \end{vmatrix}$ with respect to local frame and the center of gravity of link 2 I_{cg2} locates $P_{2c2} = \begin{vmatrix} x_{c2} \\ y_{c2} \\ 0 \end{vmatrix}$ in terms of its local frame, the new positions in terms of frame {0}, which is the world frame, are obtained by multiplication of each relating vectors up.

$$l2\_cg^{0}P = {}^{0}_{1}R \cdot P_{1c1} = \begin{vmatrix} c\theta_1 & s\theta_1 & 0 \\ 0 & 0 & -1 \\ s\theta_1 & c\theta_1 & 0 \end{vmatrix} \cdot \begin{vmatrix} x_{c1} \\ y_{c1} \\ 0 \end{vmatrix} = \begin{vmatrix} x_{c1}c\theta_1 - y_{c2}s\theta_1 \\ y_{c2}c\theta_1 + x_{c1}\, s\theta_1 \\ 0 \end{vmatrix} \quad (6)$$

$$l3\_cg^{0}P = {}^{0}_{2}R \cdot P_{2c2} = \begin{vmatrix} -c\theta_{12} & s\theta_{12} & 0 \\ 0 & 0 & -1 \\ -s\theta_{12} & -c\theta_{12} & 0 \end{vmatrix} \cdot \begin{vmatrix} x_{c2} \\ y_{c2} \\ 0 \end{vmatrix} \quad (7)$$

Where function cosine and sine satisfy the equation that c(x) = cos(x), s(x) = sin(x) and angle θ_{12} satisfies the equation that $\theta_{12} = \theta_1 + \theta_2$ respectively.

2.2.3 Kinematic of cylinders

The kinematics of cylinders is complicated to calculate due to the geometry of each cylinder and neighboring link. Therefore, triangle is formed for both cylinders to have geometric analysis. Afterwards, transformation matrixes of cylinders in their local frames are obtained with the same fashion as those of crane links. After having kinematic of cylinders and rigid booms, a series chain of booms and cylinders can be simulated in the virtual environment as form of position vector 3x1 vector.

2.2.3.1 Transformation matrixes of first cylinders

As to represent the relationship of the first link with cylinder 1, the geometry relationships between them are calculated in this section. Imaging that if the piston of cylinder is adjusted, there will be a change in angle θ_1 and thus the first link will move up or down respectively. According to the figure, a triangle is formed by cylinder 1 and another 2 fixed links. Afterwards, laws of cosine is applying, thus, some important parameters could be solved in this way.

$$\mathrm{rj}_1 = \sqrt{{r_1}^2 + {d_1}^2} \quad (8)$$

$$\mathrm{rj}_2 = \sqrt{{r_2}^2 + {d_2}^2} \quad (9)$$

$$\varphi_1 = \frac{\pi}{2} + \theta_1 - \operatorname{atan}\left(\frac{d_1}{r_1}\right) - \operatorname{atan}\left(\frac{d_2}{r_2}\right) \quad (10)$$

$$x_{p1} = \sqrt{{\mathrm{rj}_1}^2 + {\mathrm{rj}_2}^2 - 2\mathrm{rj}_1 \cdot \mathrm{rj}_2 \cdot \cos(\varphi_1)} \quad (11)$$

$$\cos(\beta_1) = \frac{{rj_1}^2 + {x_{p1}}^2 - {\mathrm{rj}_2}^2}{2rj_1 x_{p1}} \quad (12)$$

Page 17

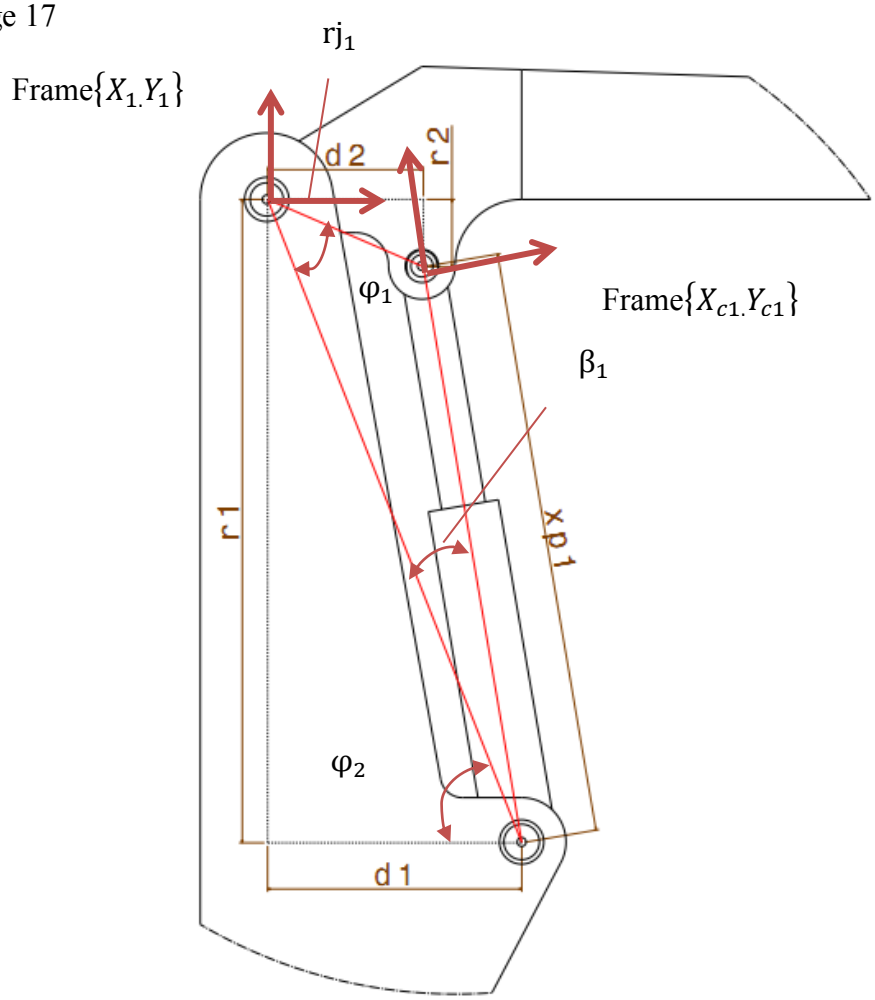


Figure 5: Geometry analysis of Joint 1

The rotation angle related to the start of cylinder frame {xc1, yc1} to the joint 1 frame{x1, y1} is$-(\varphi_1 + \theta_1)$. However, θ_1 is not shown in the figure as the joint 1 is in the initial position. Considering the fact that the crane moves in X and Y axis, the rotation matrix can be seen as the simple rotation around z axis. Furthermore, as the 1st link stays static all the time, therefore, the rotation matrix from cylinder 1 to joint 1 can be regarded to the ones from world frame directly.

$$
\begin{aligned}
{}^{0}_{c1}\mathrm{R} &= {}^{0}_{1}R \cdot {}^{1}_{c1}R \\
&= \begin{vmatrix} -c(-(\beta_1+\theta_1)) & s(-(\beta_1+\theta_1)) & 0 \\ -s(-(\beta_1+\theta_1)) & c(-(\beta_1+\theta_1)) & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} c(\beta_1+\theta_1) & -s(\beta_1+\theta_1) & 0 \\ s(\beta_1+\theta_1) & c(\beta_1+\theta_1) & 0 \\ 0 & 0 & 1 \end{vmatrix}
\end{aligned} \tag{13}
$$

Considering the position of start point of cylinder 1 locates at ${}^{1}_{c1}P \begin{vmatrix} c_{11x} \\ c_{11y} \\ 0 \end{vmatrix}$ in the frame frame{x1, y1}, the position of cylinder 1 in the common frame is:

$$ {}^{0}_{c1}P = {}^{0}_{1}T\,{}^{1}_{c1}P = \begin{vmatrix} c_{11x}c1 + c_{11y}c1 \\ c_{11x}s1 - c_{11y}c1 \\ 0 \end{vmatrix} \quad (14) $$

Where the variable c1, s1 denote the abbreviation of $\cos(\theta_1)$, $\sin(\theta_1)$ respectively.

2.2.3.2 Joint variable θ_1

After connecting θ_1 with x_{p1} geometrically, the length of cylinder stroke 1 correlates with the rotation angle 1. With the equation derived in the previous section, the correlation between parameter θ_1 and x_{p1} is graphed into figure below. The length of cylinder stroke 1 x_{p1} can be expressed in respect of angle θ_1 as following

$$ x_{p1} = \sqrt{{r_1}^2 + {d_1}^2 + {r_2}^2 + {d_2}^{2^2} - 2\sqrt{{r_1}^2 + {d_1}^2} \cdot \sqrt{{r_2}^2 + {d_2}^2} \cdot \cos(\tfrac{\pi}{2} + \theta_1 - \operatorname{atan}\left(\tfrac{d_1}{r_1}\right) - \operatorname{atan}\left(\tfrac{d_2}{r_2}\right))} \quad (15) $$

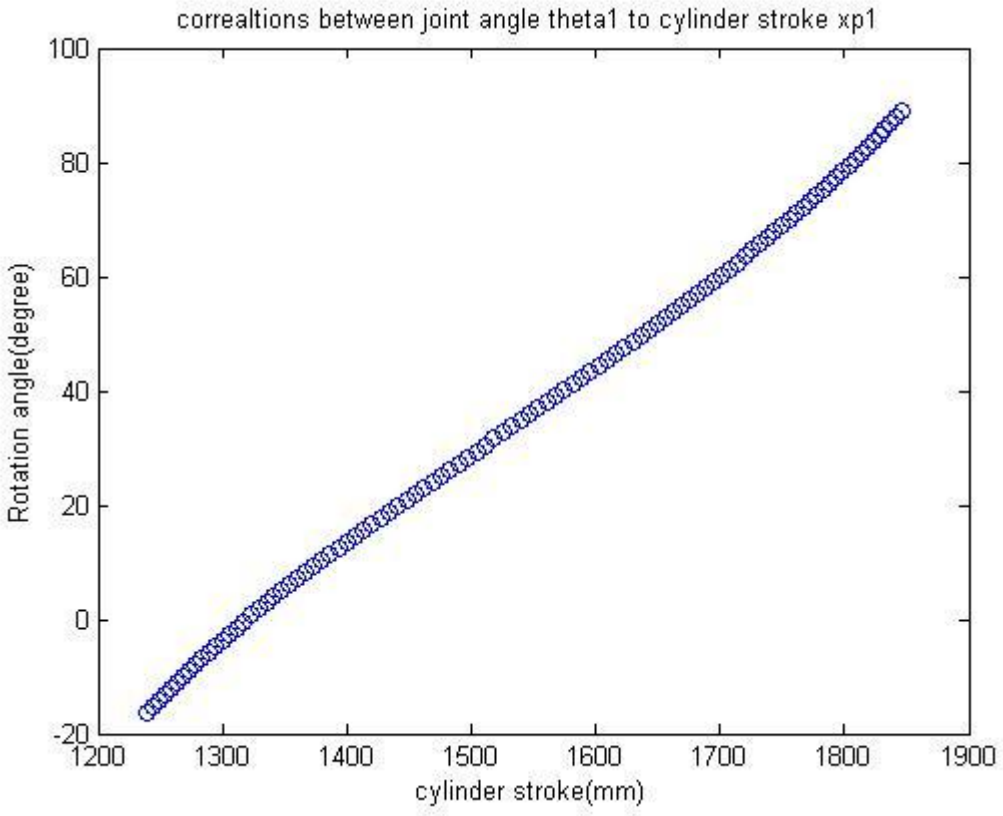


Figure 6: Proportion relationship between θ_1 and cylinder 1 stroke x_{p1}

Assuming that crane is always operating under stable situations that is angle rotates under 90° and this rotation angle 1 is not effected by the 2nd cylinder as well as torques acting on this cylinder. In this case, cylinder stroke x_{p1} perform linearly with angle θ_1. Cylinder stroke will extend to roughly 600mm, that is to say, the entire cylinder stroke will extend to approximately 1900mm when the joint angle reaches around 80. However, as the spring forces' simulation is introduced in the dynamic of

cylinders in Matlab, certain inaccuracy at boundaries of cylinders will take place. The effect will be discussed in the dynamic of crane later.

2.2.3.3 Transformation matrix of second cylinder

As for the transformation of cylinder 2, it is considered to be more complicated to calculate due to one closed-loop mechanism that is used to transfer the cylinder torque 2 to control the movement of joint 2 indirectly. The mechanism which consists of links $l_4, l_5, l_1 l_2$ plays a vital role in the analysis where several triangles are formed to find out relationship between cylinder stroke x_{p2} and joint angle θ_2. The first step is to define the angle α_3 where is the lower angle in the torque mechanism, all other parameters such as l_3, l_4, l_5 and others, could be calculated easily, and thus ,the length of cylinder piston x_{p2} and rotation angle θ_2 to cylinder frame.

$$\alpha_3 = (\text{atan}\left(\frac{r_4}{d_4}\right) + \frac{\pi}{2} + \text{atan}\left(\frac{d_5}{r_5}\right) + \theta_2) \tag{16}$$

$$l_3 = \sqrt{{l_1}^2 + {l_5}^2 - 2 l_1 l_5 \cos(\alpha_3)} \tag{17}$$

$$\cos(\beta_2) = \frac{{l_5}^2 + {l_3}^2 - {l_1}^2}{2 l_5 l_3} \tag{18}$$

$$\cos(\alpha_2) = \frac{{l_4}^2 + {l_3}^2 - {l_2}^2}{2 l_4 l_3} \tag{19}$$

$$\alpha_{xp2} = \pi - (\alpha_2 - \beta_2 - \text{atan}(\frac{r_4}{d_4})) - \text{atan}(\frac{r_3}{d_3}) \tag{20}$$

$$x_{p2} = \sqrt{{l_4}^2 + {l_6}^2 - 2 l_4 l_6 \cos(\alpha_{xp2})} \tag{21}$$

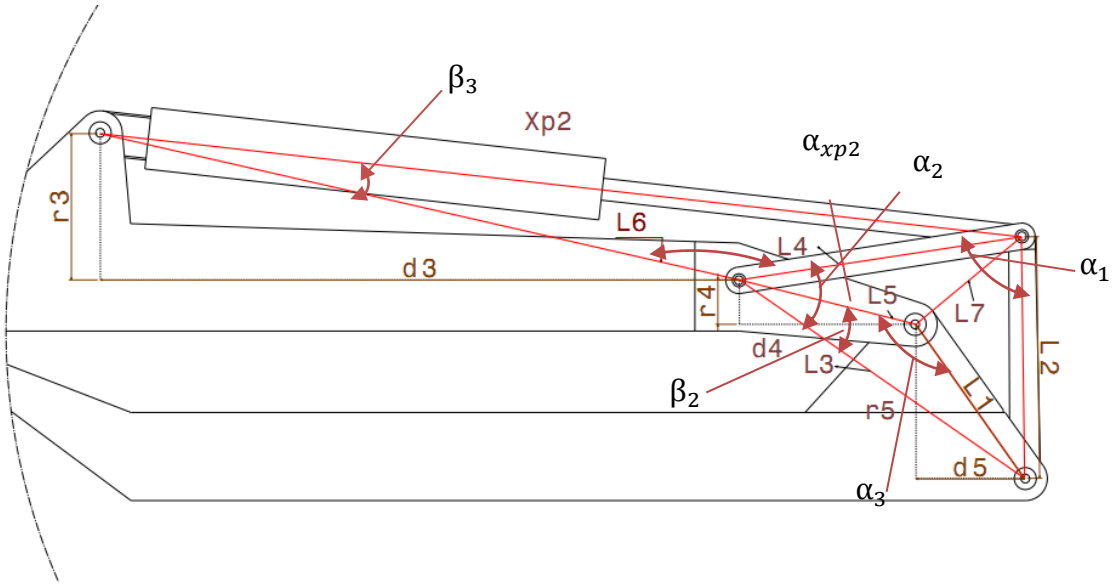


Figure 7: Geometry analysis of Joint 2

Assuming that the equation satisfies the condition that $\alpha_3 > 180°$ $\theta_2 > 50°$. Hence, just a simply case is considered. The exsiting closed-loop mechanism makes the

cylinder 2 to be more complicated than those in link 1. Calculation of l_3 is of great importance as it relates the joint variable θ_2 with torque link l_4. The next step is to calculate the roation and position matrix of cylinder frame 2 $\{X_{c2,}Y_{c2}\}$ and torque link 4 l_4 related to joint 1 frame$\{X_{1,}Y_1\}$ respectively.The vector chains to locate frames are graphed in the figure 6.

The cylinder frame 2 is the location where pressure of cylinder 2 exerts on. To be more specific , cylinder frame 2 suffers a rotation by an angle of $\beta_3 + \text{atan}\left(\frac{r_3}{d_3}\right) + \frac{\pi}{2}$ with respect to origin of joint 1.

$$ {}^{0}_{c2}\mathrm{R} = {}^{0}_{1}R \cdot {}^{1}_{c2}R = \begin{vmatrix} -c\left(\beta_3 + \text{atan}\left(\frac{r_3}{d_3}\right) + \frac{\pi}{2}\right) & s\left(\beta_3 + \text{atan}\left(\frac{r_3}{d_3}\right) + \frac{\pi}{2}\right) & 0 \\ -s\left(\beta_3 + \text{atan}\left(\frac{r_3}{d_3}\right) + \frac{\pi}{2}\right) & c\left(\beta_3 + \text{atan}\left(\frac{r_3}{d_3}\right) + \frac{\pi}{2}\right) & 0 \\ 0 & 0 & 1 \end{vmatrix} \quad (22) $$

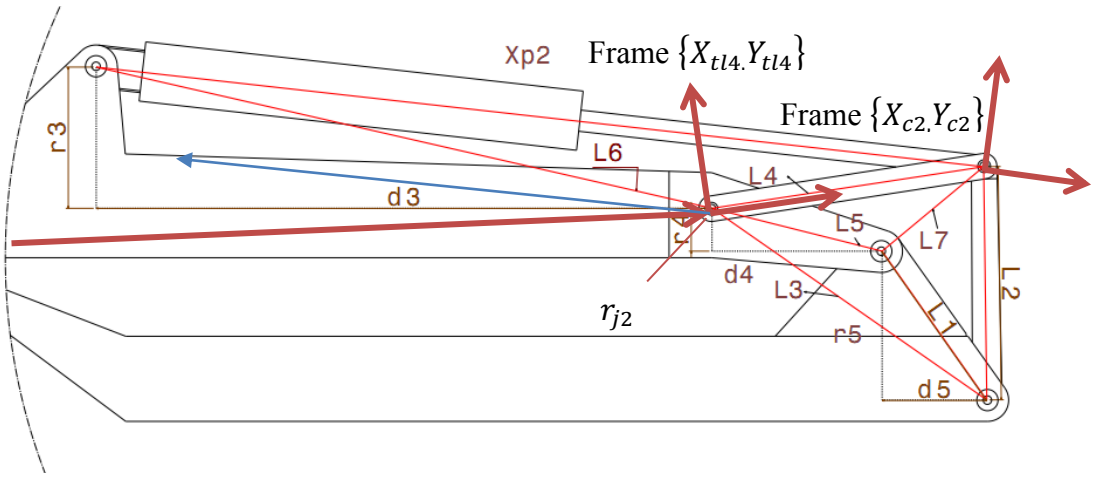


Figure 8: Vectors' chains to locate frame$\{X_{tl4,}Y_{tl4}\}$ and $\{X_{c2,}Y_{c2}\}$ in respect with joint 1 frame $\{X_{1,}Y_1\}$

Assuming that start of link l_4 with respect to joint 1$\{X_{1,}Y_1\}$ locates at $\begin{vmatrix} x_{l4} \\ y_{l4} \\ 0 \end{vmatrix}$ and start of cylinder 2 P_{tl4} with respect to frame $\{X_{tl4,}Y_{tl4}\}$ locates at $\begin{vmatrix} l_4 \\ 0 \\ 0 \end{vmatrix}$. Finally, the position of cylinder 2 can be computed with the help of vectors' chain is,

$$ {}^{0}_{c2}P = {}^{0}_{1}R \cdot \begin{vmatrix} x_{l4} \\ y_{l4} \\ 0 \end{vmatrix} + {}_{tl4}^{1}\mathrm{R} \begin{vmatrix} l_4 \\ 0 \\ 0 \end{vmatrix} \quad (23) $$

Where rotation matrix ${}_{tl4}^{1}\mathrm{R}$ from torque l_4 to the joint 1 can be further stated as equation below

$$ {}^{1}_{tl4}\mathrm{R} = {}^{0}_{1}R \cdot {}^{1}_{tl4}R = \begin{vmatrix} -c(\beta_2 + \alpha_2 - \operatorname{atan}\left(\frac{r_4}{d_4}\right)) & s(\beta_2 + \alpha_2 - \operatorname{atan}\left(\frac{r_4}{d_4}\right)) & 0 \\ -s(\beta_2 + \alpha_2 - \operatorname{atan}\left(\frac{r_4}{d_4}\right)) & c(\beta_2 + \alpha_2 - \operatorname{atan}\left(\frac{r_4}{d_4}\right)) & 0 \\ 0 & 0 & 1 \end{vmatrix} \quad (24) $$

The result of corresponding position and orientation of two cylinders in detail are displayed in the Matlab *mainprgram.m* scripting shown in the Appendix B.

2.2.3.4 Joint variable θ_2

It is more complicated to express the internal relationship between θ_2 and corresponding cylinder stroke x_{p2}. Then, length of cylinder stroke 1 x_{p2} can be expressed in respect of angle θ_2 as following

$$ x_{p2} = \sqrt{{l_4}^2 + {l_6}^2 - 2l_4 l_6 \cos(\pi - (\operatorname{acos}(\tfrac{{l_4}^2 + {l_3}^2 - {l_2}^2}{2l_4 l_3}) - \operatorname{acos}(\tfrac{{l_5}^2 + {l_3}^2 - {l_1}^2}{2l_5 l_3}) - \operatorname{atan}(\tfrac{r_4}{d_4})) - \operatorname{atan}(\tfrac{r_3}{d_3}))} \quad (25) $$

Where parameter l_3 can be further developed as ,

$$ l_3 = \sqrt{{l_1}^2 + {l_5}^2 - 2l_1 l_5 \cos(2\pi - (\operatorname{atan}\left(\frac{r_4}{d_4}\right) + \frac{\pi}{2} + \operatorname{atan}\left(\frac{d_5}{r_5}\right) + \theta_2))} \quad (26) $$

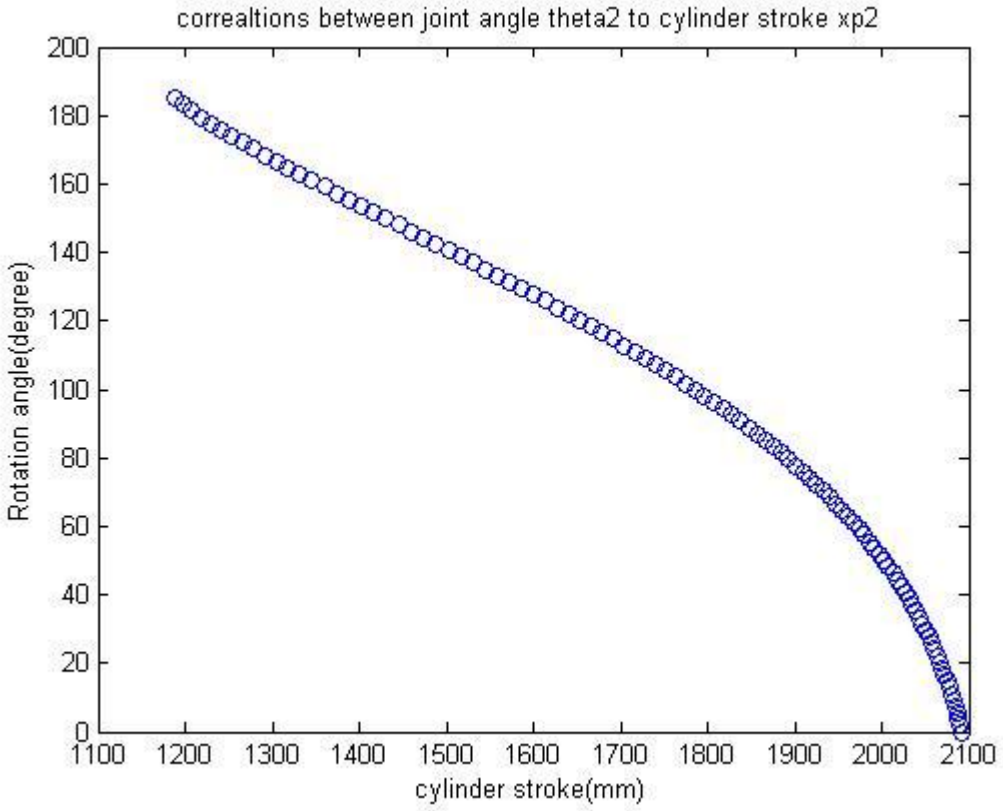


Figure 9:Proportion relationship between rotation angle θ_2 and cylinder stroke x_{p2}

Assuming that crane is always operating under stable situations that is angle rotates under 180° . Geometrically, the cylinder stroke x_{p2} is only effected by roation angle θ_2 . However, the angle is impacted by joint angle θ_1 dynamically due to the torque from cylinder 1. Hence, correlation in between joint angle θ_1 and x_{p2} would be taken out of concern. With angle θ_2 reaches maximum value at around 180° , cylinder 2 stroke x_{p2} retracts to about 1800mm. In this case, cylinder stroke x_{p2} and joint angle θ_2 are almost linear after the cylinder stroke x_{p2} exceeds 400mm.

2.2.4 Crane Workspace

The workspace of crane denotes the volume of space that the end-effector of this manipulator can reach. In this case, because the 1st link is static base, there is no rotation around Z axis for the whole manipulator. In this sense, the workspace area is the total area that has been passed by the crane tip as far as it can reach in the X - Y plane. As the working area is constrained by 2 joints, the limitation of cylinder stroke x_{p1},x_{p2} will be determinded first and thus, the resulting oint angle θ_1, θ_2 can be summerized in the following table. Users need to notice that limitations of crane parameter are calculated without considering the forces that causes the motion.

Table 2: Limitation of crane joints and extension of cylinders.

Parameters	Maximum value	Minimum value
θ_1	$-16.32°$	90.13°
θ_2	0°	182.13°
x_{p1}	1240mm	1850mm
x_{p2}	1189mm	2105.41mm

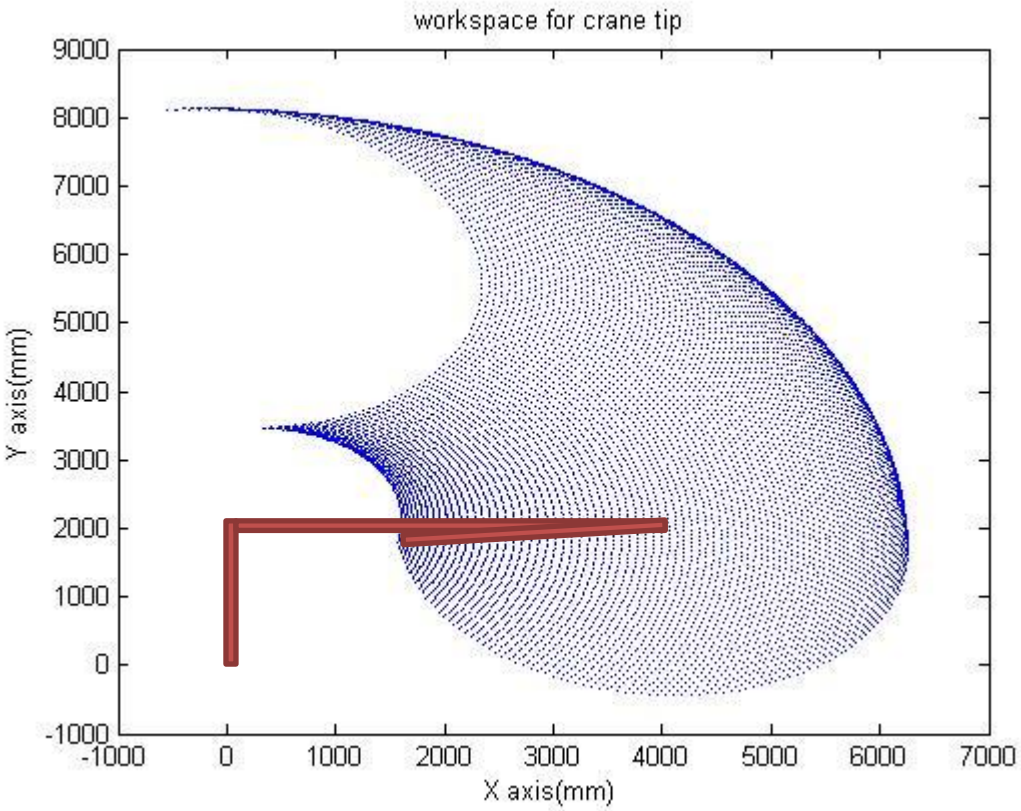


Figure 10: *workspace of crane with rough model of crane*

With the limitations of joint variables and cylinder strokes, it is possible to graph the working area of crane according to table 2. The three rectangles in red represent the crane structure 1st link, 2nd link and 3rd link roughly and the area in blue dashes represents the reachable place of the crane tip when two cylinders operate inside their physics boundaries geometrically. Users also need to bear in mind that further constraints will be developed due to obstacle avoidance as well as safety tolerance and spring force will be used to slow down the crane-tip before leaving its working area.

2.3 Dynamic Equation of Crane

The derivation of a dynamic of crane plays an important role in simulation of crane. Unlike kinematics of crane which is excluded the consequence of external forces that cause the motion, dynamics of crane describes the motion of crane with external forces and torques. Therefore, this section will consider an approach to solve dynamic motion of crane into ordinary differential equation in Matlab.

2.3.1 Lagrangian Formula

The dynamic formulation provides a means of deriving the equation of motion from a scalar function, called Lagrangian, it is defined as the difference between the kinetic and potential energy of a mechanical system. (J.Cragic, 2005) Since the Lagrangian Formula is straight forward entirely, there is no need to consider joint forces

respectively. Generally, the Lagrangian equation ($L(q,\dot{q})$) of a manipulator can be found as

$$L(q,\dot{q}) = k(q,\dot{q}) - u(q) \tag{27}$$

Where parameter k is the total kinetic energy and u is the total potential energy of the system respectively, parameters $q, \dot{q}$ denotes the joint variable vector and its derivation. Note that the number of degree of freedom (DOF) has an effect on the vector size. In this case, system has 2 DOF, that is to say, crane vectors will have 2 components and 2 variables. The generalized Lagrangian equation can be displayed as:

$$\frac{d}{dt}\frac{\partial k}{\partial \dot{q}} - \frac{\partial k}{\partial q} + \frac{\partial u}{\partial q} = \tau \tag{28}$$

Where τ can be given as the external torques obtained from cylinders associated with q.

2.3.2 Jacobian Matrix

Jacobian is a linear transformation which can be used to relate vector of joint angle to its Cartesian velocities. The number of rows equals the number of degree of freedom. Moreover, joint torques can be linked with static force by Jacobian in force domain. (J.Cragic, 2005) The Jacobian matrix of each link and cylinder in planar can be obtained by taking derivation of corresponding position associated with joint variable vector or by Euler approach. In these approaches, the Jacobian matrix satisfies the equation $\mathrm{v} = \mathrm{J}(\mathrm{q})\dot{q}$ or $\tau = \mathrm{J}(\mathrm{q})^T \mathrm{f}$ respectively. It is possible to calculate angular velocity and torque easily with the help of Jacobian matrix. On the other hand, the vector of coordinate q describes the position and orientation of links much more efficiently than getting Jacobian matrix from Euler approach. In our case, the Jacobian matrixes are computed as,

$$J_{vi}(q) = \frac{\partial P_{cgi}}{\partial q} \tag{29}$$

Where q satisfied the equation $\mathrm{q} = \begin{bmatrix}\theta_1 \\ \theta_2\end{bmatrix}$, $J_{vi}(q)$denotes the Jacobian Matrix and P_{cgi} implements the ith position vector of center of gravity of each boom which has been calculated and represented in the Kinematics of crane chapter.

2.3.3 Kinetic Energy

The kinetic energy consists of two kinds of different energy. The first term is rotational energy due to linear velocity of the link's center of gravity and second term is translational one due to the angular velocity of the link's center of gravity. Therefore, the total kinetic energy can be expressed as a summation of each kinetic energy stored

in each link. The entire kinetic energy satisfies,

$$K = \sum_{i=1}^{n} k_{i_i} = \frac{1}{2} m_i {v_{cgi}}^T v_{cgi} + \frac{1}{2} \ {}^i w_i^T \, {}^{Cg_i} I_i \ {}^i w_i \tag{30}$$

In the equation above, the parameter m_i denotes the total mass of the ith link in the crane, velocity $v_{cgi}\ and\ {}^i w_i$, refers to the linear and angular velocity at center of gravity related and parameter ${}^{C_i} I_i$ refers to the inertia tensor on the center of gravity of each boom.

Next, focus will be put on solving the unknown parameters in the equation (30) , that is the linear v_{cgi} and angular velocities ${}^i w_i$ of the center of each boom given as a function associated with the world frame. The linear velocities v_{cgi} are introduced in terms of Jacobian matrix as mentioned in the Jacobian matrix section.

$$v_i = J_{ci}(q) \cdot q \tag{31}$$

Whereas, the angular velocities ${}^i w_i$ at center of gravity can be considered as the rotation round Z axis and thus expressed as

$$\omega_1 = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{bmatrix} \tag{32}$$

$$\omega_2 = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 + \dot{\theta}_2 \end{bmatrix} \tag{33}$$

Finally, by replacing the equation with v_i, ω_1, ω_2, the new kinetic energy of each link yields in

$$K_i = \frac{1}{2} m_i \left(\frac{\partial A_{ci}}{\partial q} \cdot \dot{q}\right)^T \frac{\partial A_{ci}}{\partial q} \cdot \dot{q} + \frac{1}{2} \ {}^i w_i^T \, {}^{C_i} I_i \ {}^i w_i \tag{34}$$

Where A_{ci} implementing position vector of center, ${}^{C_i} I_i$ is inertia tensor of each boom respectively.

2.3.4 Potential Energy

Single potential energy that is stored in the system takes place due to gravitational forces. In addition, entire potential energy is always the sum of energies in the individual links at the center of gravity. The entire potential energy satisfies the equation (35),

$$U = \sum_{i=1}^{n=2} u_i \ = m_i \ {}^0 g^T \ {}^0 P_{C_i} \tag{35}$$

where ${}^{0}P_{C_i}$ denotes the position of each boom at center of gravity and g is the gravity vector due to gravitational forces. In this case, ${}^{0}P_{C_i}$ vectors can be expressed easily with DH transformation matrixes associated with joint variables at equation (6) and (7), detail information can be referred to Appendix B.

2.3.5 External Torques

The external torques τ_i and its resulting force f_i are motivated by the pressure actuated in the cylinders due to internal hydraulic pressure and valves operation. However, as the hydraulic control algorithm is not considered in this paper, one should assume that pressure p in each cylinder is static and not varied with time and external torque only exerts on the start of each cylinder.

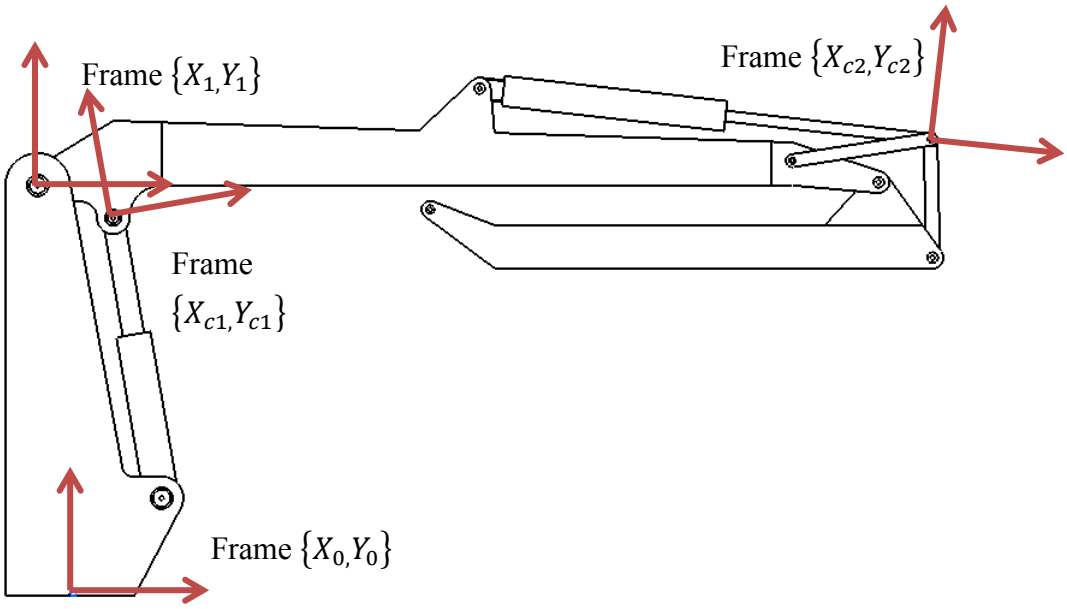


Figure 11: Evaluation of generalized forces frame

In figure 11 sketches the cylinder frame of each torque. Because orientation matrix at the point of cylinder origin has been computed in the kinematic part, the vector τ stands for the torques that is applied onto crane links in equation (36). By separating two cylinders from the mechanical part; the corresponding cylinder force can be regarded as external forces f acts onto the crane boom.

Hereby, Jacobian matrix is introduced again to transpose Cartesian forces which relate forces to its local joint into equivalent joint torques in one common frame. The entire external torque is the summary of each external torque.

$$\tau_i = {}^{0}J^{T}\,{}^{0}f_i$$
$$= \sum_{i=1}^{m} {}^{i}_{0}R^{T} \frac{\partial P_{cy_i}}{\partial q}^{T} f_i \qquad (36)$$

Where force parameter f_i denotes external force vectors related to the inertial frame,

positions P_{cy_i} denote the position of each cylinder in their coordinates. With the multiplication with rotation matrix ${}^{i}_{0}R^{T}$, it is possible to transfer all external torque into world frame.

2.3.6 Equations of motion

The generalized Lagrangian Equation will be further derived and categorized according to mass $M(q)$, velocity $Q(q,\dot{q})$ and torque τ sections respectively. By replacing the parameter in equation (37), a new Lagrangian Equation is given in the equation (38).

$$\frac{d}{dt}\frac{\partial k(q,\dot{q})}{\partial \dot{q}} - \frac{\partial k(q,\dot{q})}{\partial q} + \frac{\partial u}{\partial q} = \tau \tag{37}$$

One should notice that the new Lagrangian Equation is based on the ideal model, which means that there are other effects which have been ignored, such as bending effects of rigid bodies due to large forces and other friction ratio that may take place at joints.

By the distributive law and time derivative for the partial derivate related to joint variable q, equation yields to

$$\frac{\partial}{\partial \dot{q}}\left(\frac{\partial k(q,\dot{q})}{\partial \dot{q}}\right)\ddot{q} + \frac{\partial}{\partial q}\left(\frac{\partial k(q,\dot{q})}{\partial \dot{q}}\right)\dot{q} + \frac{\partial}{\partial t}\left(\frac{\partial k(q,\dot{q})}{\partial \dot{q}}\right) - \frac{\partial k(q,\dot{q})}{\partial \dot{q}} + \frac{\partial u(q)}{\partial q} = \tau \tag{38}$$

By replacing main components into simple notation as

$$\mathrm{M(q)} = \frac{\partial}{\partial \dot{q}}\left(\frac{\partial k(q,\dot{q})}{\partial \dot{q}}\right) \tag{39}$$

$$\mathrm{Q(q,\dot{q})} + G(q) = \frac{\partial}{\partial q}\left(\frac{\partial k(q,\dot{q})}{\partial \dot{q}}\right)\dot{q} + \frac{\partial}{\partial t}\left(\frac{\partial k(q,\dot{q})}{\partial \dot{q}}\right) - \frac{\partial k(q,\dot{q})}{\partial \dot{q}} + \frac{\partial u(q)}{\partial q} \tag{40}$$

Rewriting and simplifying the generalized Lagrangian Equation into this following:

$$M(q)\ddot{q} + Q(q,\dot{q}) + G(q) = \tau \tag{41}$$

By moving the terms related to vectors $\mathrm{q},\dot{\mathrm{q}}$ and M to the right side of equation, the general acceleration of dynamic equation could be solved in the closed form as shown below.

$$\ddot{\mathrm{q}} = \mathrm{M}^{-1}(\mathrm{q})[\tau - \mathrm{Q(q,\dot{q})} - \mathrm{G(q)}] \tag{42}$$

Where q satisfied the equation $\mathrm{q} = \begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}$ for two rotational angles θ_1, θ_2 $\dot{\mathrm{q}}, \ddot{\mathrm{q}}$ denotes the angular velocities and acceleration of two joints.

Finally, the establishment of state space model is of great importance as the equation regarding $\ddot{\mathrm{q}}$ is the base of Matlab simulation in the ODE function. According to Wikipedia, state space (abbreviated SS) model is the core of Mathematic model of a

physical system as a set of input, output and state variables related by first-order differential equation. To abstract from the number of inputs, outputs and states, the variables are expressed as vectors. In this case, the entire state space model can be expressed as

$$\begin{bmatrix}\dot{q}\\ \ddot{q}\end{bmatrix} = \begin{bmatrix}\dot{q}\\ \mathrm{M}^{-1}(\mathrm{q})[\tau - \mathrm{Q}(\mathrm{q},\dot{\mathrm{q}}) - \mathrm{G}(\mathrm{q})]\end{bmatrix} \tag{43}$$

2.4 Initial conditions of the crane

2.4.1 Initial conditions of cylinder forces

The initial value of forces is able to lift crane to any position that is reachable by basic torque computation. Assuming that Link i has mass of m_i and its center of gravity locates at l_{ci}. External force f_i situates at l_{fi} with respect to joint 1. The entire crane can be represented in the following torque equations:

$$f_1 l_{f1} = m_1 g l_1 + (m_2) g l_c \tag{44}$$
$$f_2 l_{f2} = m_2 g l_2 \tag{45}$$

Thus, if all the parameters are taken in from crane parameters into formula, the equation regarding f_1, f_2 yield to

$$f_1 \cdot 0.31 = 4280 \cdot 2.1 + (2390) \cdot 2.9$$
$$f_2 \cdot 0.36 = (2390) \cdot 1.1$$

Finally, $f_1 = 51353N$, $f_2 = 7302N$ (46)
should be provided in the least case if crane keeps static dynamically during simulation.

2.4.2 Equilibrium point of holding the crane

The equilibrium point of a system is achieved when the system is in balance and thus the corresponding state vectors of state space model are zero at this moment. In this case, crane will be in the equilibrium point if state vectors, angular velocities and accelerations stay at zero.

$$\begin{bmatrix}\dot{q}\\ \ddot{q}\end{bmatrix} = \begin{bmatrix}\dot{q}\\ \mathrm{M}^{-1}(\mathrm{q})[\tau - \mathrm{Q}(\mathrm{q},\dot{\mathrm{q}}) - \mathrm{G}(\mathrm{q})]\end{bmatrix} = \begin{bmatrix}0\\ 0\end{bmatrix} \tag{47}$$

Therefore, the cylinder forces needed for holding the crane in initial positions are calculated based on equation above as:

$$f_1 = 138430N\,, f_2 = 36347N$$

The related plots on angular displacement of joints are shown in the Chapter 6 of comparisons on result part.

3 Modeling crane in Blender Game Engine

3.1 Introduction to Game Physics and Blender Game Engine

3.1.1 Introduction to Game Physics Engine

Game physics or so-called computer animation physics involves the introduction of the laws of physics into a simulation or game engine, particularly in 3D computer graphics, for the purpose of making the effect appear more real to the observer. Typically, simulation physics is only a close approximation to real physics, and computation is performed using discrete values. (Wikipédia) The physics functions existed in the game engine supports particle effects such as real explosion, wind effect, flight simulation, car physics, crates, destructible objects, cloth and ragdolls along with many other comprehensive effects. (Whyte, 2002)

Game engine and collision detection are two main components in game physics. Game physics engine is a reusable simulator or a big calculator which computes mathematics equation needed to simulate physics and abstracts away from particular projectiles with specific of each game's scenario in order to simulate the general physics. However, the game engine does not 'recognize' the object of simulation and therefore, providing the engine with the proper physical characteristics and employing correct physics function on objects are both of great importance to have a realistic simulation.

Collision detection is a sensor which detects if any components or objects are touching each other. In other words, in order to make sure that any area of space cannot be occupied by more than one object, collision detection should be employed to ensure reliability and make simulation more accurate. (3D Theory- Collision Detection)

3.1.2 General introduction to Blender R2.6

Blender is a free and open-source 3D computer graphics software product used for creating animated films, visual effects, interactive 3D applications or video games. Its high quality 3D graphics set similar with other commercial high-end 3D software such as Maya software, Cinema 4D. It is able to model, render, simulate and animate objects in various scenes in Blender. Commonly, Blender is used to design quality and realistic scenes for various video games and 3D movies such as movie Toy Story released by PIXAR. Recently, thanks to refine development made by its physics engine, Blender is turning into a powerful dynamic simulator in real world compared to its previous

versions. (Blender Game Engine Overview)

Existing user interface in Blender is displayed in figure 15. The general interface of Blender could be divided into 3 parts; on the top, there is the INFO window, it displays Blender version, information about the Blender file to present Blender's main menu. The other 2 parts are 3D windows and bottom windows both enlightened in orange. A bottom window contains most of the useful tools for performing material or physics functions shown in small icons. For example, Object editor could help relocate position or orientation of active object, Material editor for editing surfaces, textures, and much more.

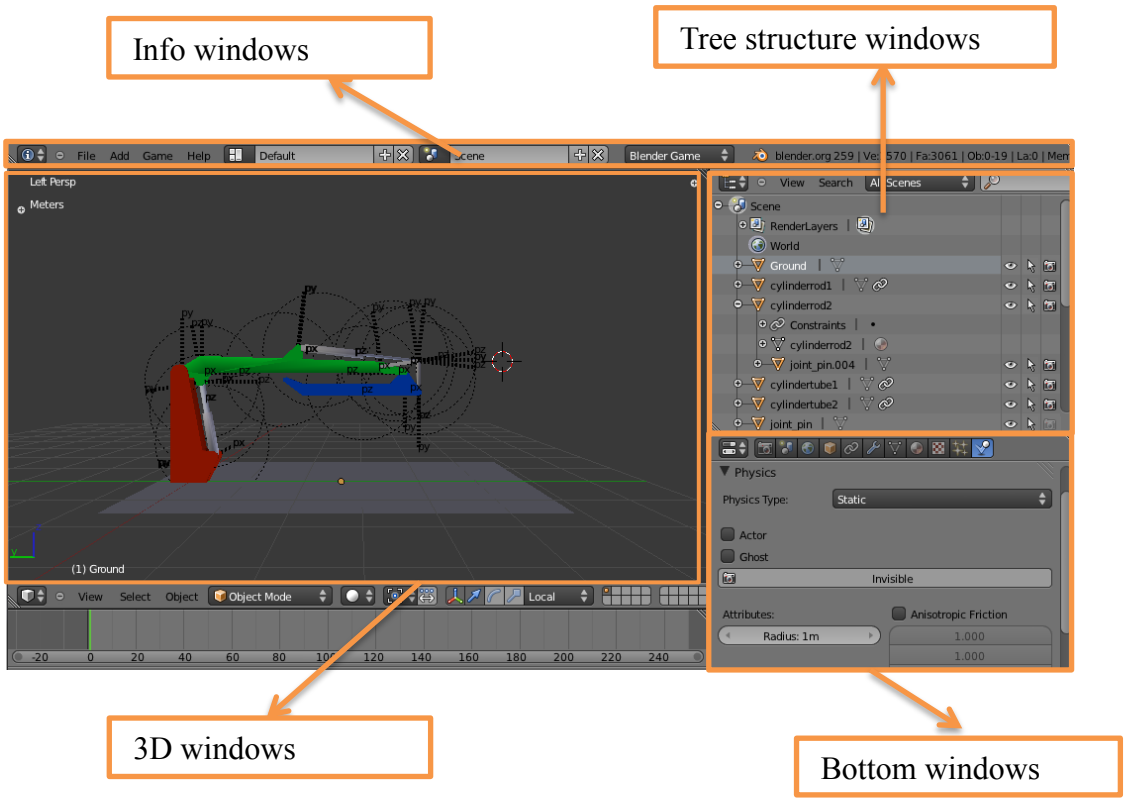


Figure 15: General user interface of Blender R2.5.

In addtion, Blender could also be seen as a software that integrates functions with programming notions in a mathematical perspective. The philosophy of Blender design consists of three main tasks: storage of data, edition and visualization. Any 3D object may be represented by an array of veritices as a mesh. Users may operate on any sub parts of the array to modify material or other properties of a mesh.

With clear division and definition of each window in Blender, it is easier to make change on objects interiorly or exteriorly.

3.1.3 Blender game engine (BGE)

Blender has its own built-in game engine that allows users to create interactive 3D applications freely. It is a powerful programming tool which mainly concentrates on development of realistic simulation actions for video game. The engine is the first game

engine on which games can be developed without any programming skills required because it uses a click-and-drag visual interface to list all the scripts. (Roosendall T,Wartmaan C, 2003) In the click-drag interface, BGE uses a system of graphical 'logic bricks' which is a combination of 'sensors', 'controllers' and 'actuators' to control the movement and display of objects in the engine. (Blender Game Engine Overview)

Generally, the physics engine is one of the basic components required to develop physics-based simulation or virtual environment. (J R Juang,W H Hung,S C Kang, 2010) Additionally, Blender game engine can be extended to different areas via Python scripts for high level scripting developer, which make it possible for users to control the game objects in the scene by Python controllers as well.

Simulation are composed of incremental changes spanning multiple frames, it is taken frame by frame. The key frames define the value of data at specified frame. Between these key frames, the properties' values are computed by Blender and filled in. The converter between Blender key frames to real time unit is about thirty key frames to one second in real time during simulation.

3.1.4 Blender Physics

Game physics is the core factor to make all kinds of simulation act in real-time in Blender. The Blender engine utilizes Bullet Physics as its official Physics application. Bullet Physics is an open source physics engine featuring 3D collision detection, soft body dynamics as well as rigid body dynamics.

Among three features, soft body dynamics is a field of computer graphics that focuses on visually realistic physical simulation of the motion or properties on deformable objects or soft bodies. The shape of soft bodies can change under any circumstance. There is a wide range of applications of soft body such as cutting of soft bodies, tearing of cloth or plasticity or wind effect etc.

Unlike soft body, rigid bodies occupy space and need to have geometrical properties such as center of mass and moments of inertia that characterize motion in six degree of freedom. Contrast to deformable material of soft body, material deformation function does not have a significant effect on the motion of the system. (Bullet (software) in Wikipedia) With the help of Bullet Physics, the physics engine allows objects to fall due to gravitational forces, roll and collide with other objects in more realistic manners. In this way, it increases interactivity and realism in game.

3.1.5 Python Scripts and its API

Python is an interpreter, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level of dynamic data types, and classes as external library files of Blender. Python combines remarkable power with very clear syntax. It was expressly designed to be usable as an extension language for applications that need a programmable interface. Python is probably the single part of Blender which achieved the greatest improvement. A full reorganization of what existed was carried out and many new modules added in each newly-released of

Blender version. (Bullet (software))

Application programming interface (API) of Python is the interface that the program library provides to support requests for services made by the computer programmers. (Chi H L, Hung W.H,Kang S.C, 2007) Python is used by C and C++ programmers who want to write extension modules to explore other challenging tasks that are needed to accomplish by game objects. (Python/C API Reference Manual)

3.2 Development of Crane simulation in Blender

This section shows entire establishment of physical crane model in Blender game engine. Several physical features provided by game engine are introduced to render a crane model to act like a real one. The entire process of developing the crane simulation can be separated into three main steps as shown in the process flow according to figure 16: (1) loading 3D model of crane, (2) physics feature setting, (3) game logic setting. Those steps allow users to use straightforward commands to execute simulation.

Figure 16: Overall working consequence of crane simulation in Blender

During the process for setting up crane simulation in Blender, the first step is to import ready-made crane model into Blender software. Afterwards, all the necessary physics and dynamics properties of crane model will be employed in detail during this section. As the final step, the crane is able to react properly with external forces acting on crane cylinder achieved by logic controllers called 'logic bricks', more details illustration of logic brick will be introduced in section 3.4.

3.2.1 3D modeling of crane

The basic crane model with detailed dimensions is created in Solidworks and imported into Blender afterwards. Basic crane model has three main components,which are supporting links, joints and two actuating cylinders.

There are two revolute joints connectting three major links, so that crane is able to reach different positions in the planar space. Meanwhile, there are other revolute joints used to connect torque links with cylinders. Following shows the main structure of creane, different components with features are varies with different colors .

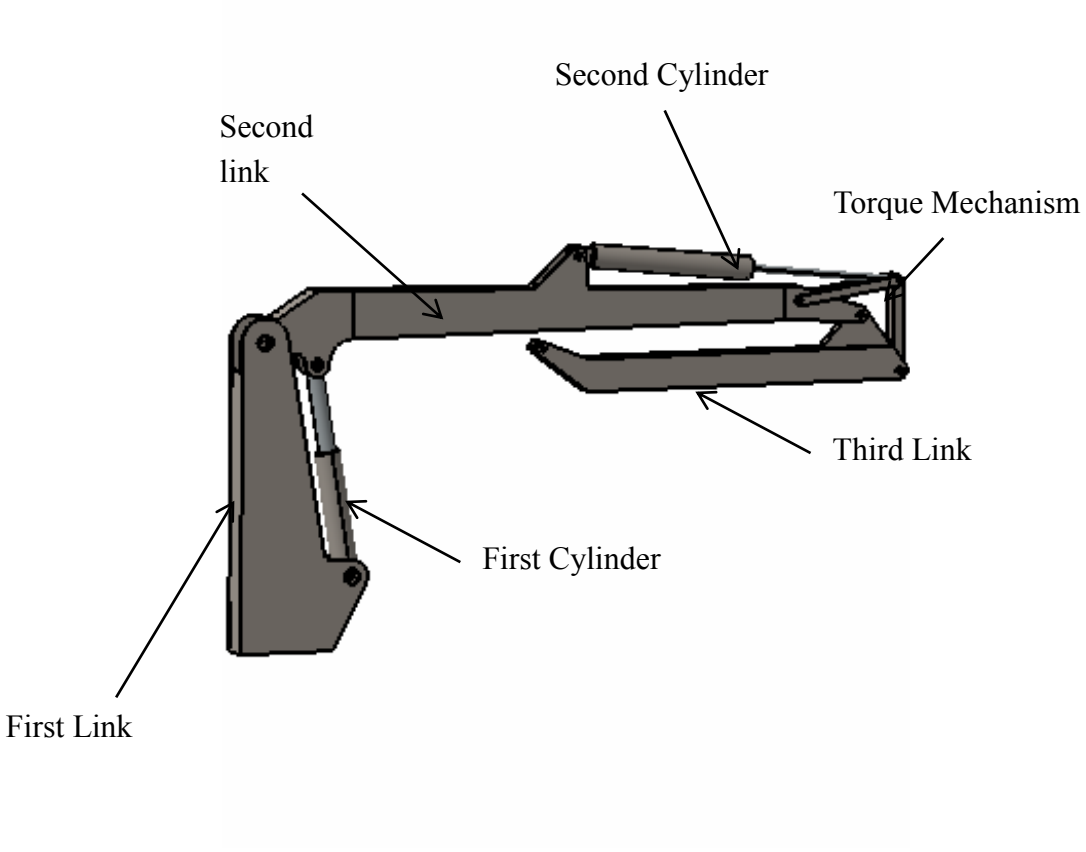


Figure 17: 3D model illustrations of crane

The following notes clarify different working tasks for links, joints and cylinders.

- ✧ First link acts as a base and would be static in simulation and other two revolute joints connect in between three links.
- ✧ Two actuating cylinders get pressure inputs from hydraulic pumps and these inputs are the general inputs to activate the whole crane.

3.2.2 Overview of physics features of crane model

Generally, main purpose of setting up physics feature is to construct crane model as multi-rigid bodies so that each component is physically connecting with its neighboring links. For instance, if certain external force is applied to one boom, the force could easily effect on neighboring booms as if crane booms affect each other in real life. The entire process to build crane dynamics can be separated into several main steps as shown in the process flow: (1) setting up basic parameters for general environment, (2) setting up physics properties of crane components, (3) developing physics joints between crane elements.

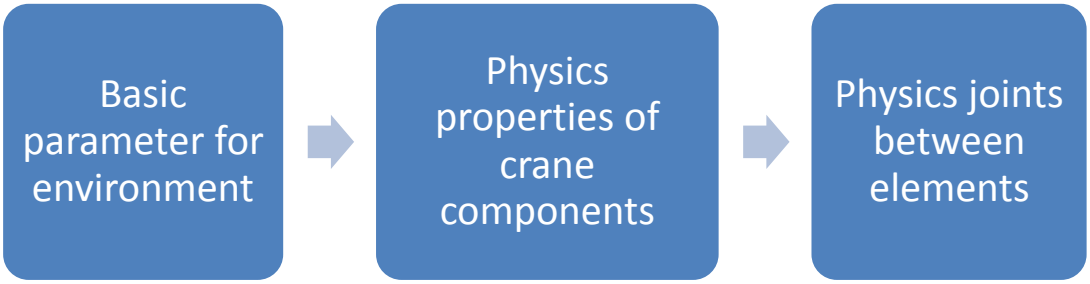


Figure 18: 3D model illustrations of crane

3.2.2.1 Physics environment for scene

The physics properties include the physics requirement for generating the general environment, such like types of physics engine, general gravitational forces, and unit of measurements in game engine.

One example of those settings is shown on how general physics environment and physics engine are determined in the Blender environment. The settings for general environment is a basic and essential step because it standardizes the physics generally so that all the other physics features for different individual items could be set based on uniform standard.

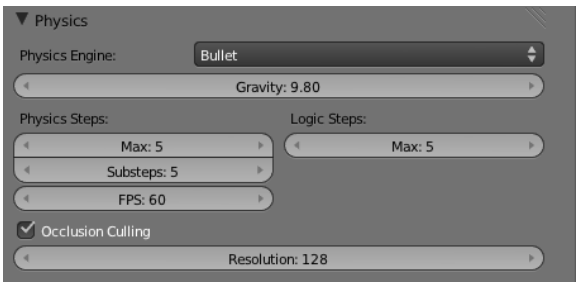


Figure 19: Selection of game engine and physics steps.

3.2.2.2 Object types, inertia tensors and collision bounds settings

(a) Rigid body types

Physics parameters denote individual physics properties of different components such as mass, object density and physics type. Two panels shown in the figure 19 tells where users define all objects properties. Among all properties, physics type is quite essential property for each component, because this feature tells game engine which kind of object is simulating and its related characteristics.

There are several main types of physics to be selected, i.e. occlude, no collision, sensor, static, dynamic, rigid body and soft body. In a finite-element model, certain relatively stiff parts can be represented by rigid bodies when stress distributions and wave propagation in such parts are not critical. An advantage of rigid bodies rather than deformable finite elements is computational efficiency. Elements that belong to the rigid bodies have no associated internal forces or stiffness. For instance, rigid body type is a common selection for object which is solid, while soft body is type of object that will

deforms on collision. In this research, only the base of crane is selected to be static physics because its base is fixed to ground while others are rigid body types.

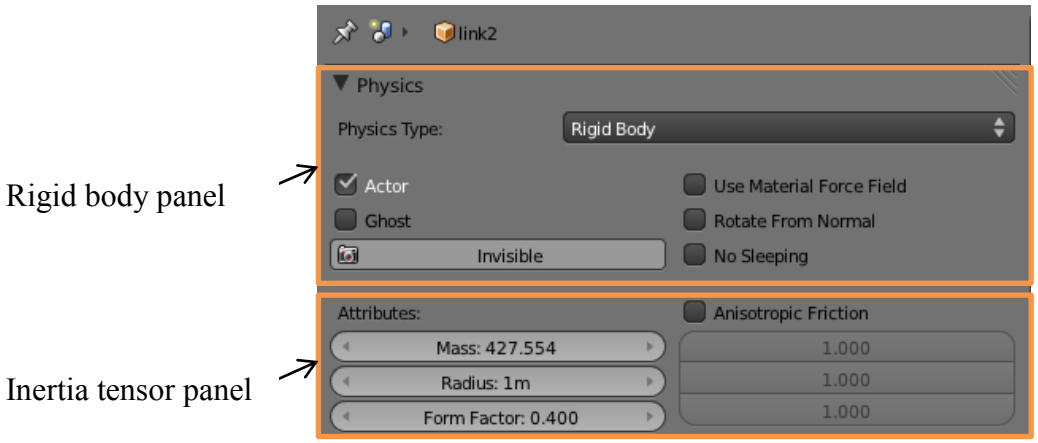


Figure 20: Object type menus in the Properties panel

(b) Inertia tensors

The inertia sensor is a measure of object's resistance to changes to its rotation like the mass to linear motion. It is the inertia of a rotating body with respect to its rotation. The moment of inertia of an object about a given axis describes how difficult it is to change its angular motion about that axis. Therefore, it encompasses not just how much mass the object has overall, but how far each bit of mass is from the axis. The further out the object's mass is, the more rotational inertia the object has, and the more torque (force* distance from axis of rotation) is required to change its rotation rate. (Wikipédia)

Radius feature listed in the figure 20 checks for the radius of bounding sphere originated at center of existing object, where the sphere represents range of inertia tensor for this object. While, form factor scales the mass inertia tensor scales from a unit matrix to zero inertia tensor.

There is nearly no possible way to modify the inertia tensor to the exact values that users want as those have been pre-defined by Blender. However, it is possible to form factor in Blender as a compensation of inertia tensor. For example, if form factor becomes 0, it means that the mass is regarded as a point mass, while values between 0.0 and 0.4 means that mass distribution is less or more even all the way to shape's edges. So, value 0.4 means that whole length from mass origin to collision shape edge is evenly distributed. If users select values between 0.4 and 1.0, it means that the collision shape's mass distribution is concentrated towards its outer edges. If users use sphere shape analogue and assume using form factor value of 1.0, it means that your metal sphere is totally hollow i.e. shell, all mass is concentrated to the surface of the sphere.

The only way to display inertia sensor of one object is to print out the local inertia by Python scripts.

(c) Collision bounds

Page 36

After decided the physics type, collision bound of each rigid body needs to be determined to speed up computation of game physics. Game engine offers several shapes of bounds for users to select such like box, sphere and etc. Collision shapes simplifies the object being simulated to avoid the high level of processing power that it would demand. Therefore, it is often faster to use a simple form such as a box or a sphere when an object's shape is very similar.

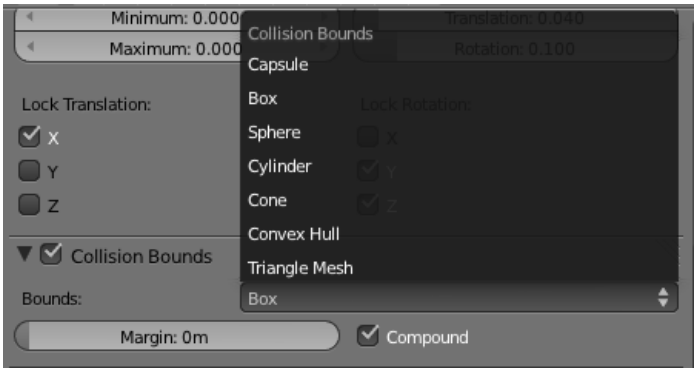


Figure 21: Object type menus in the Properties panel

3.2.2.3 Object constraints for physics joints in crane model

From Robotics point of view, physics joints connect components by implementing forward kinematic method that is one approach to link location of different components and display these in the matrix pattern filled with joint angles with respect to uniform coordinator.

Unlike complicated computation based on original forward kinematic method, physics joints are easily achieved by defining object constraints feature in game physics and consequently, game physics will automatically recognize the crane as dynamic multi-bodies which means any movement made by one component could easily affect other neighboring links.

In order to clarify the physics of joint mechanism and help setting up object constraints, one type of notation regarding physics model elements will be introduced according to report (Hung W.H, Kang S.C, 2009) displayed in table 3. This notation mainly focuses on showing different locations and types of joints other than displaying the whole structure of crane model. Physics joints alongside with the connection of each component of crane will be determined with representations of prismatic or revolute relationship between each object displayed in figure 19.

Table 3: The description of physics model elements (J R Juang,W H Hung,S C Kang, 2010) (Hung W.H, Kang S.C, 2009)

Physics element	Symbol	Description
Actor		An actor is a representation of a rigid body which visualizes a solid body of finite size that is not deformable.

Revolute Joint		A revolute joint has a single rotational degree of freedom from two objects. The axis along which the two bodies may rotate is specified with a point and a direction vector.
Prismatic Joint		A prismatic joint allows a relative translational movement between two bodies along an axis without relative rotational movement at all. It is usually necessary to add limits to joints in order to prevent the bodies from getting too far from each other along the joint axis.
Joints in a fixed distance		A solid line that connects two actors denotes the static connection and thus there is no degree of freedom between two actors which must be rigid bodies.

When displaying the crane model according to different roles in table 3, all crane links are actors minimizing to static blue dots, hence, different types and orientations of physics joints in tube or cylindrical symbols becomes rather easy to notice.

As shown in the figure below, there are eight rotational joints, including four joints connecting main links and other four associating cylinders and torque mechanism with major booms, while only two prismatic joints exist to link tubes and rods of cylinders. Establishment on correct connection between links and cylinders make it easier for users to get real-time simulation on crane model.

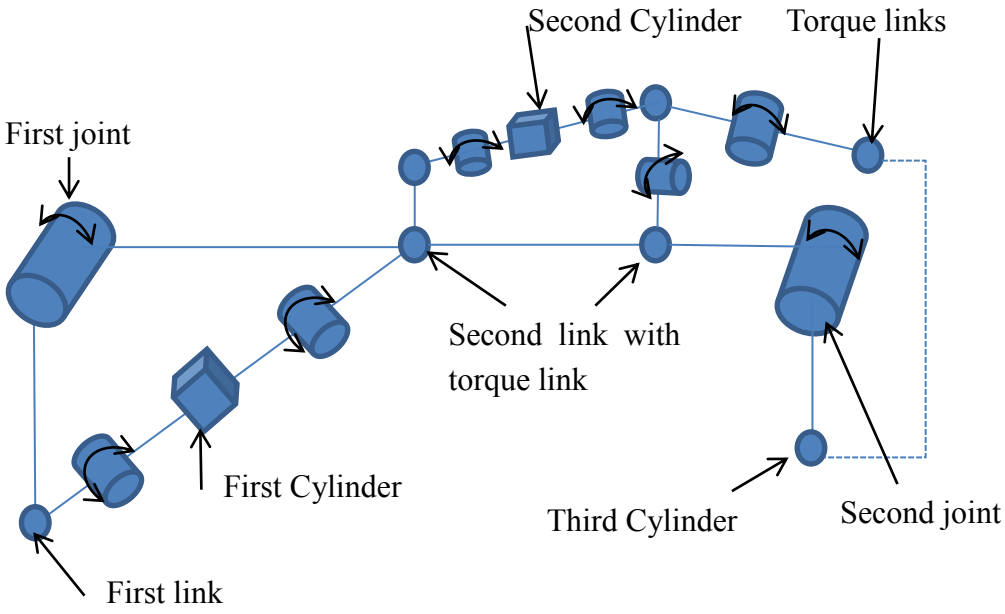


Figure 22: Physics model of a crane regarding revolute and translational joints

a) Joint connection between links

Object constraints function is usually set for mastering object features, i.e. these features control in various styles of the objects' transform properties such as position, rotation and scale. Commonly, majority of object constraints are made for game render to locate, rotate or scale objects automatically since there is no game physics provided in game render animation.

Despite most constraints intended for game render, users still are allowed to simulate various physics connections with one general joint constraint named rigid body joints. A rigid body is a kind of material property for object whose motion is determined by a maximum of six degrees of freedom (DOFs) with respect to pivot point of object. It is widely used for numerical simulation of multi-bodies dynamic applications via joint elements as well as connected to flexible bodies to model mixed rigid-flexible body dynamics.

The constraints panel shown in the figure 23 is a sub-context parent to the Object context. Generally, rigid body joints consist of three basic types; hinge, ball-and-socket and generic joints.

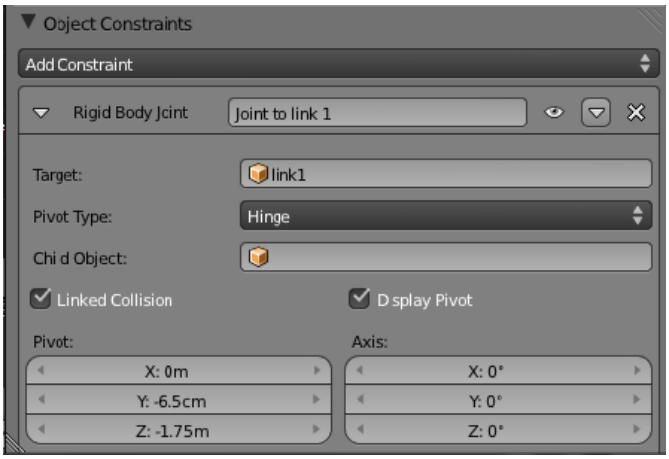


Figure 23: Cylinder tube objects Constraint panel

The hinge joint or revolute joint is designed for objects who have one degree of freedom, the target object which is the object that is subjective to the rotation and translation of the owners just rotates around X axis of its pivot point and thus any kind of movement on Y, Z axis is restricting to none. It is widely used to connect links by a revolute joint in game engine. For example, second link can be connected to base by a hinge joint and thus it is able to rotate around this hinge joint.

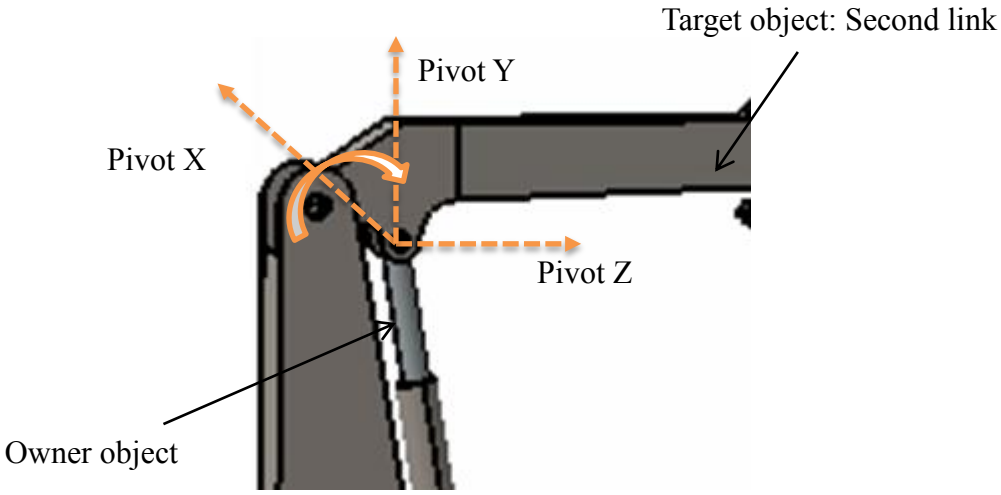


Figure 24: The Pivot position of hinge joint around x axis on links.

b) Joint modeling between links and cylinders

The revolute joints between links and cylinders work in the same fashion as modeling connection between main links. Hinge joints connect cylinders and links together, so cylinders will attach to links and rotate on local x-axis.

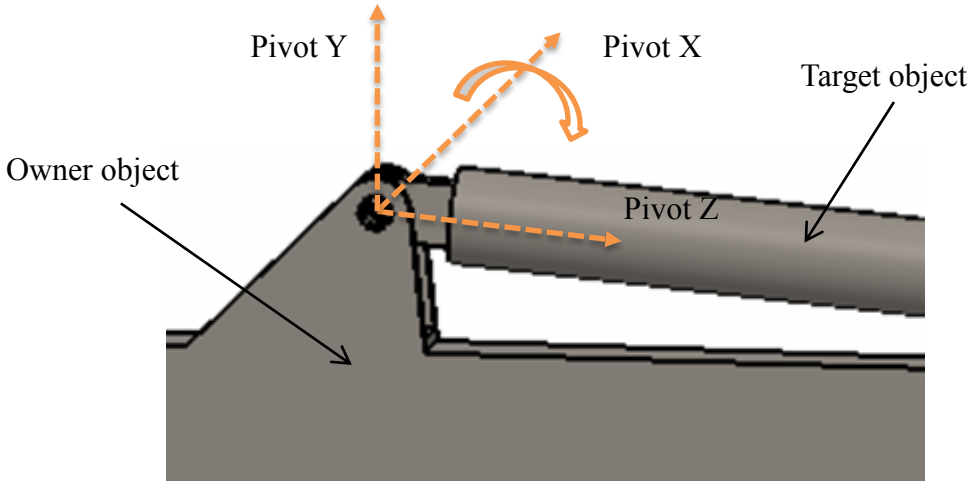


Figure 25: The Pivot position of hinge joint around x axis for cylinder tube.

In this way, all basic joint connections of crane components are achieved by hinge joints easily.

c) Physics modeling of cylinders

As mentioned in previous section, generic joints provide users with a special tool to emulate customize joints that are not pre-defined in rigid body joints by configuring each of the 6 degrees of freedom. Compared with other types of joints in rigid body joints, the target object of generic joint is no more constrained at a fixed distance and orientation from the pivot point of owner, it is the reason why it is also can be used to

simulate prismatic joint in particular.

When applying this tool to crane model, generic joints are used in defining two translational joints to simulate cylinder motion. Unlike common prismatic joints, the cylinders joints are complicated joint mechanism including cylinder rod, tube and other two links as well in figure 21. In this complicated joint mechanism, movement of each component is dependent on each other and thus it requires more two physics constraints than common revolute joints. The complex physics modeling regarding cylinder are explained in detail into separate two sessions, physics for cylinder tube and rod.

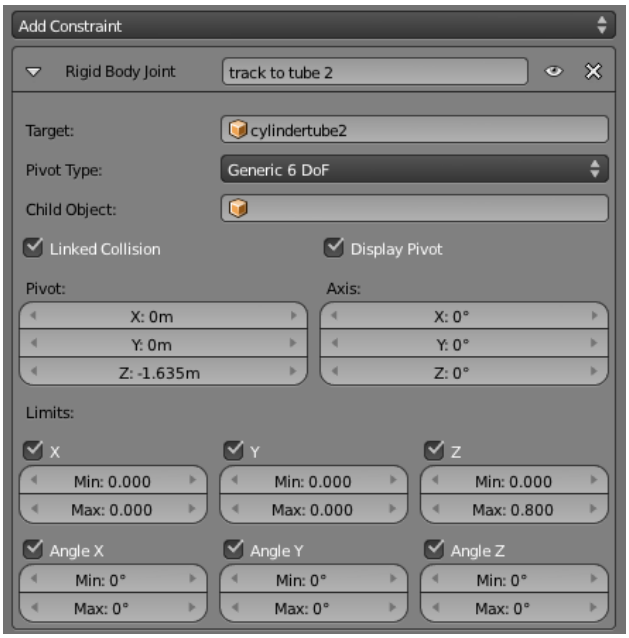


Figure 26: Setting up cylinder simulation by generic 6 DOF type for cylinder rod.

As displayed in figure 27, cylinder rod is situated in between its corresponding cylinder tube and one of torque mechanism to third link. The generic joint constraint defines the translational movement in such a way that cylinder rod can rotate around its pivot point with a changeable distance.

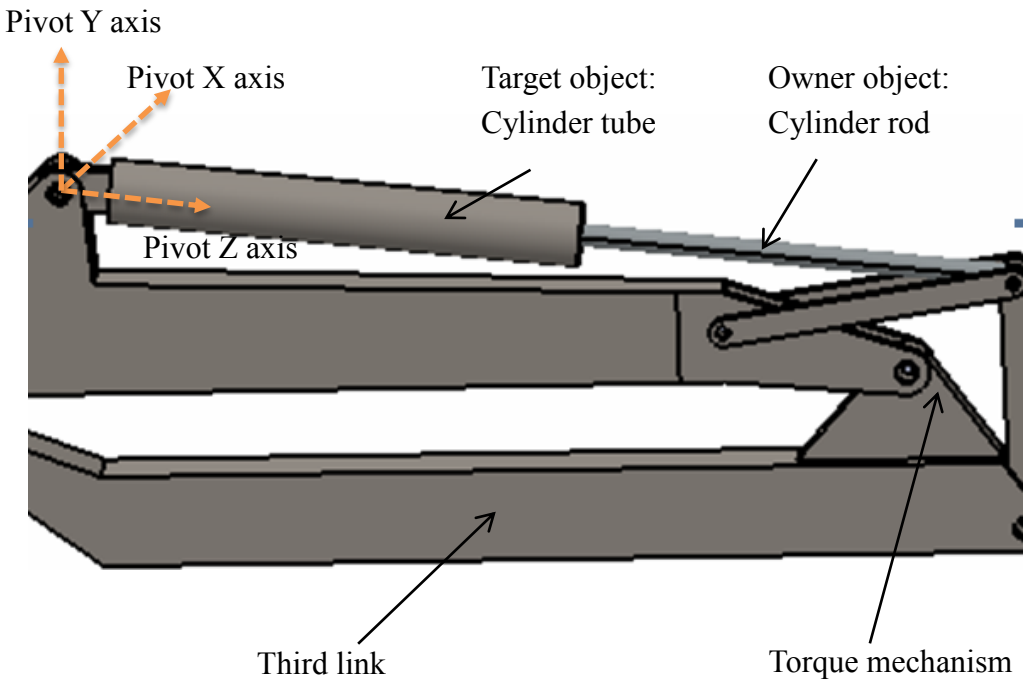


Figure 27: The customized generic joint connects cylinder rod with its corresponding tube.

Therefore, the rod has two degree of freedom, one for translational movement moving along its Z axis and one rotational movement following pivot point. As pivot point rotates along the X axis, cylinder rod will follow its rotational movement. Meanwhile, cylinder rod is allowed to move along Z axis of pivot point. In addition, because cylinder tube is a subjective object to rod, tube only follows the rotation of cylinder rod. And thus cylinder rod is able to adjust its displacement on account of the entire mechanism including cylinder tube and other links which connects to rod.

Limits function shown in figure 26 in rigid body sets limitation of movement for cylinder rod, the min and max value are computed with respect to pivot point. Cylinder rod just moves along local Z axis with respect to cylinder tube, hence, any values regarding Z axis in the generic joint is set for cylinder's movement.

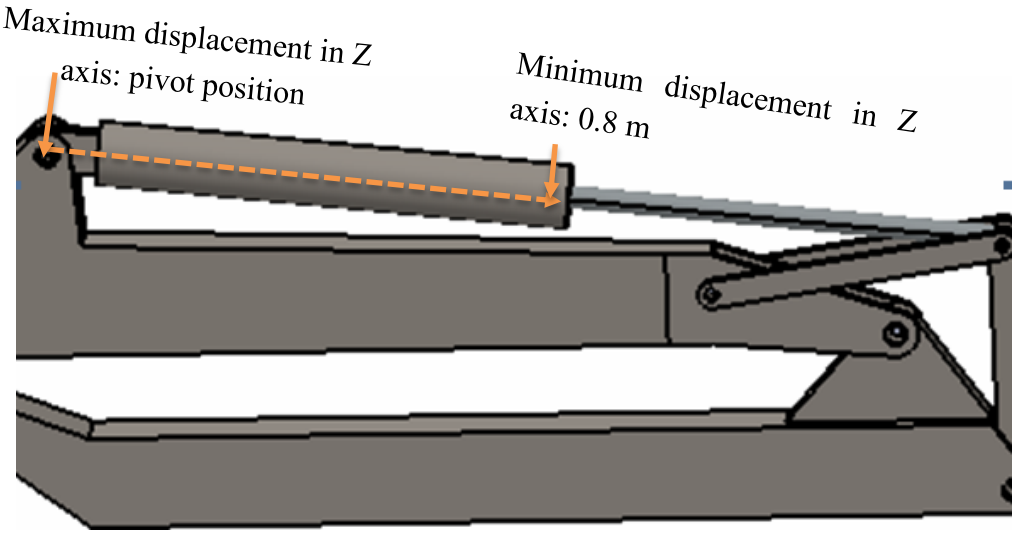


Figure 28: Setting up displacement limitation of cylinder rod.

Displacement settings of cylinder rod in figure 28 indicate that the cylinder rod can move freely from 0 to 0.8 m with respect to its pivot point. It starts at position that is 0.8 meter far away from pivot point and will stop at the pivot position in Z axis. In this way, users can simulate displacement limits for X, Y axis based on how many degree of freedom one object has.

Similarly, same genertic joint applies to its cylinder tube as well. Cylinder tube jus have one degree of freedom for revolute movement on mechanism link and no movement is allowed on local Z asix as cylinder tube is fixed to its joint. As soon as the pivot point rotates along X asix, cylinder tube follows the rotation of pivot point. Because rod is a subjective object to tube, cylinder rod will adjust its movment along with cylinder tube. On the other hand, cylinder tube is also fixed to the second link by one revolute joint, the limits on its rotation depends on displacement limitations on cylinder rod.

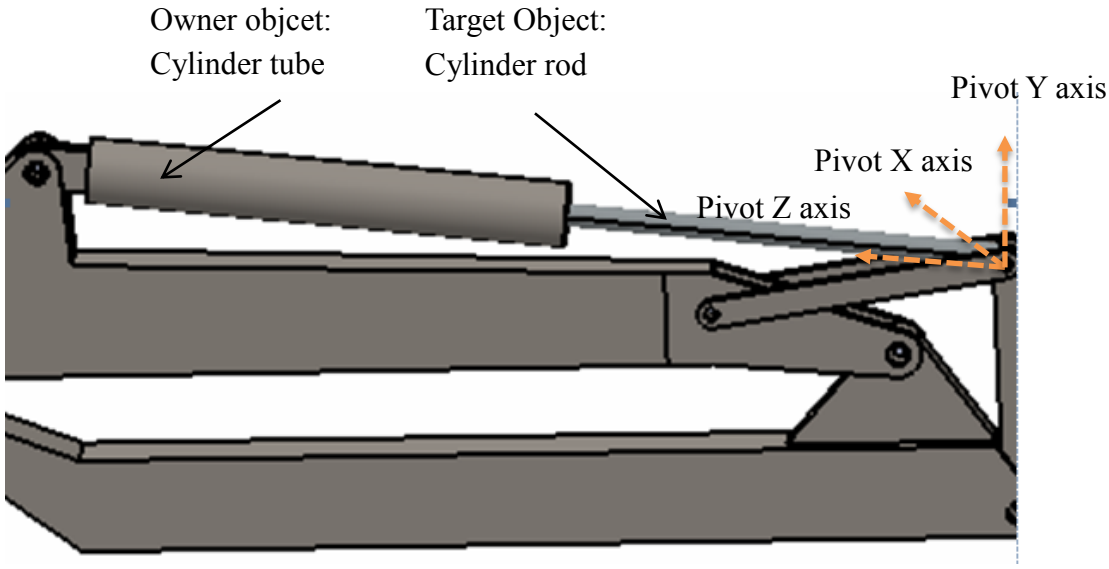
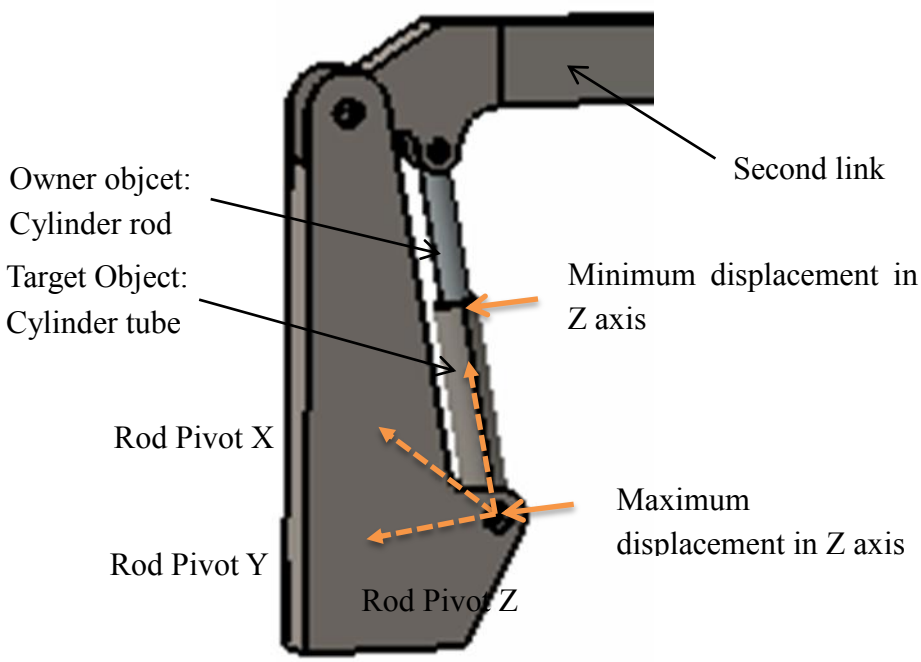


Figure 29: Pivot point of cylinder tube, it aims at rod end to track each other.

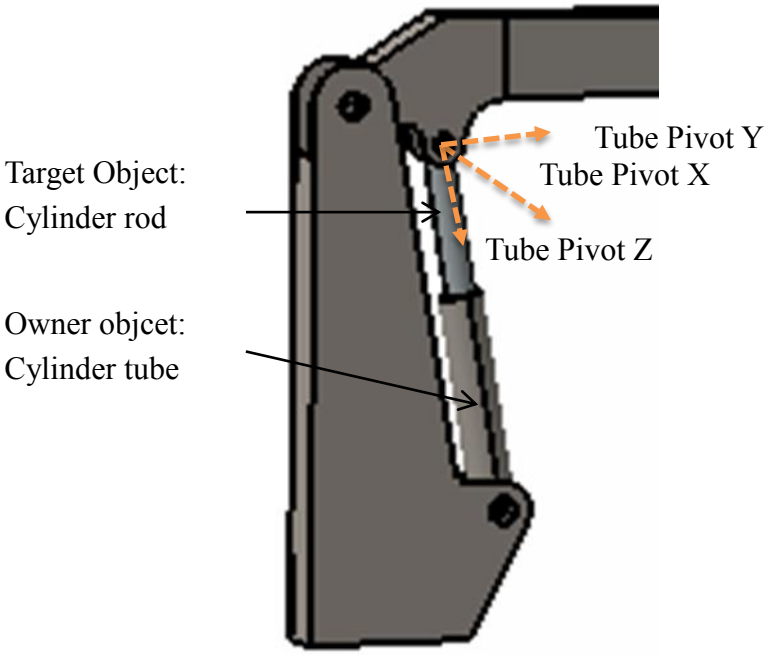
During simulation, two constraints are running simultaneously so that cylinder tube and rod target and follow movement of each other according to the pressure inputs. Included the revolute joints linking cylinder tube and second link as well as cylinder rod with toque mechanism, the cylinder could react to pressure inputs obtained from pump and lift the third link as expected.

d) Physics modeling of first cylinder

The physics modeling of first cylinder is implemented in the same fashion as the second cylinder. As displayed in figure 30, first cylinder forms a complex joint mechanism with second link and crane base. Any movement of second link is mainly triggered by the first cylinder.



(a) Pivot point of first cylinder rod, it aims at tube end to track each other.



(b) *Pivot point and limits of first cylinder tube*

Figure 30: Setting up physics modeling of first cylinder

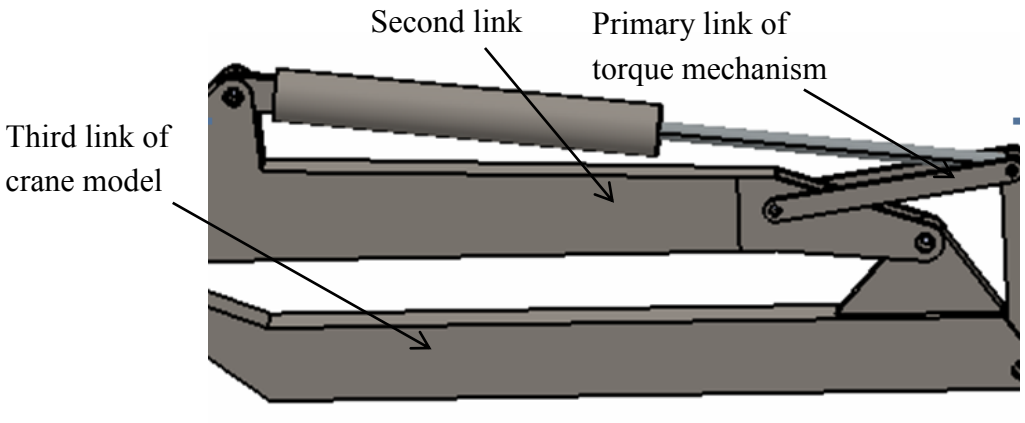
In addition, since rod is able to move along its Z axis, the start and end of cylinder tube becomes the displacement limitation of first cylinder. And thus, the limits feature in generic joint panel can be set according to cylinder tube length. Moreover, as first cylinder activates the movement of second link, the rotation displacement of first joint depends on first cylinder as well.

e) Physics modeling of torque mechanism

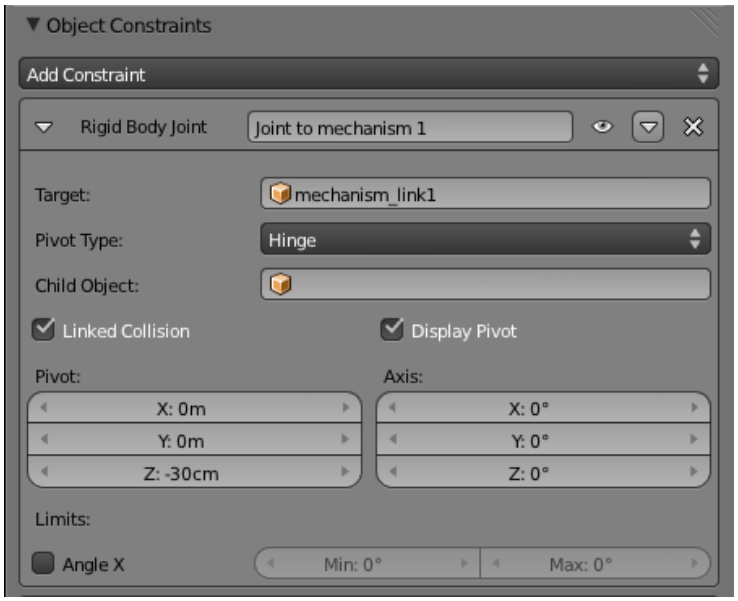
Page 45

A torque mechanism is a rectangular mechanism formed by three small links and connecting second cylinder with the third link in the crane model. It is implemented to transfer the cylinder movement to lift third links other than situating the second cylinder at the end of second link. In this sense, this torque mechanism is totally dependent on the second cylinder and thus only follows the movement from second link.

The torque mechanism can be simply linked with each other by several hinge joints.



(a) Modeling of torque mechanism



(b) *Hinge joint to simulate torque mechanism*

Figure 31: Hinge joint to simulate torque mechanism

It is important to clarify the owner and target object of each hinge joint to control the third link by jogging second cylinder. The first link is the primary link of the entire mechanism as its movement can be directly activated by cylinder rod. The second link of torque mechanism is the followers of primary one, which can be modeled in the same fashion as the primary links.

Page 46

As a conclusion, the physics modeling of joints and cylinders are of great importance because procedure sets up necessary physics features to declare physics relationships between neighboring objects. After proper establishment of all physics features in game physics, controlling algorithm based on control bricks are introduced in next session.

3.2.2.4 Setting up collision bounds and its detection

The collision detection is one of most indispensable and useful features in game physics since it is near impossible to build and detect any physical collision based on physics equations in a traditional programming way. Unlike complex computation that traditional programming brings about, game physics offers several shapes of collision bounds to choose, including static, convex hull, cone, cylinder, sphere and box shapes. Moreover, identification of objects' shapes to general types, such as boxes, or cone, hull, could help to accelerate the processing time of whole simulation.

a) Physics visual-aid feature

Collision bounds are employed into game physics as a plug-in feature to monitor collision bounds and physics joints of objects and thus eliminate any potential crashes between neighboring components during simulation. By visualizing the collision bounds of objects, users can ensure that game physics applies right physics bounds and physics joints onto right meshes.

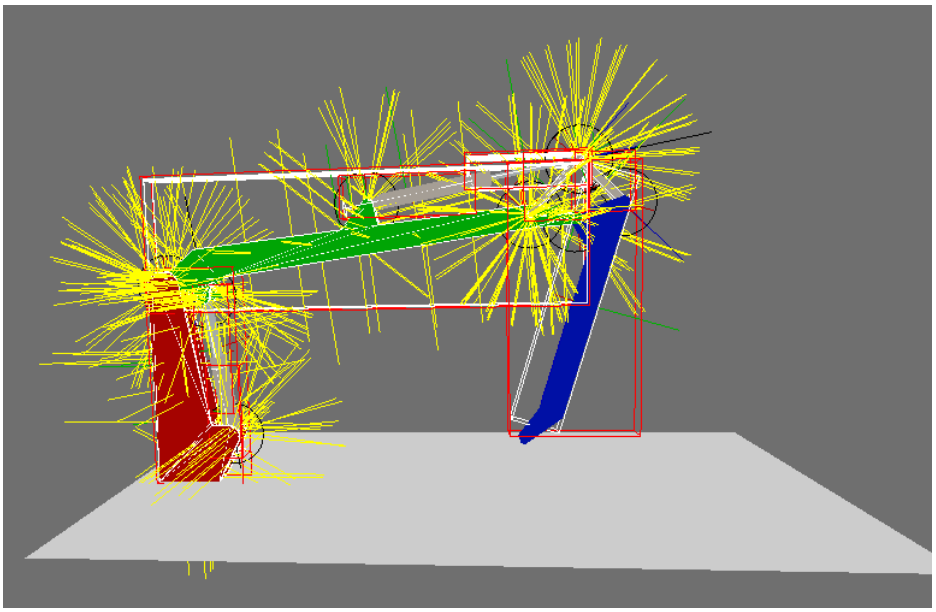


Figure 34: The physics visualization for the collision with boxes polypore

Figure 33 represents the visualized collision shape of crane model in game physics. The different types of physics are highlighted in different colors. For example, the splitting yellow lights indicate the object is a body without collision between other objects. Bounding boxes in red indicate that those objects are dynamic or rigid bodies that will

take collision into account. As mentioned in the Blender modeling, most of crane components are rigid bodies displayed in red. As for the second link in the crane, mesh triangle is used because of its irregular shapes and to avoid inner collision with pins displayed in the splitting yellow color.

During the simulation, when physics is calculating on the structures, existing objects are displayed covered with box bounds in white. The objects turn to be in blue denotes those components are still moving. When crane reaches a state of equilibrium, the objects turn to be in green denoting that objects are 'sleeping'. At the end, the entire crane will stop moving and turn into blue boxes. Thus, the efficiency of simulation will increase and reduces the possibility of jitter for motionless objects. (Tony Mullen,Erwin Coumans, 2008) In general, game physics offers a small collision margin to improve the reliability and performance of collision.

3.2.3 Game Logic setting and Python scripts

As displayed, this game logic interface is designed for users who use Python and game logic quite often since two function occupies. Therefore, Python scripting area and game logic area occupy most of space in the interface.

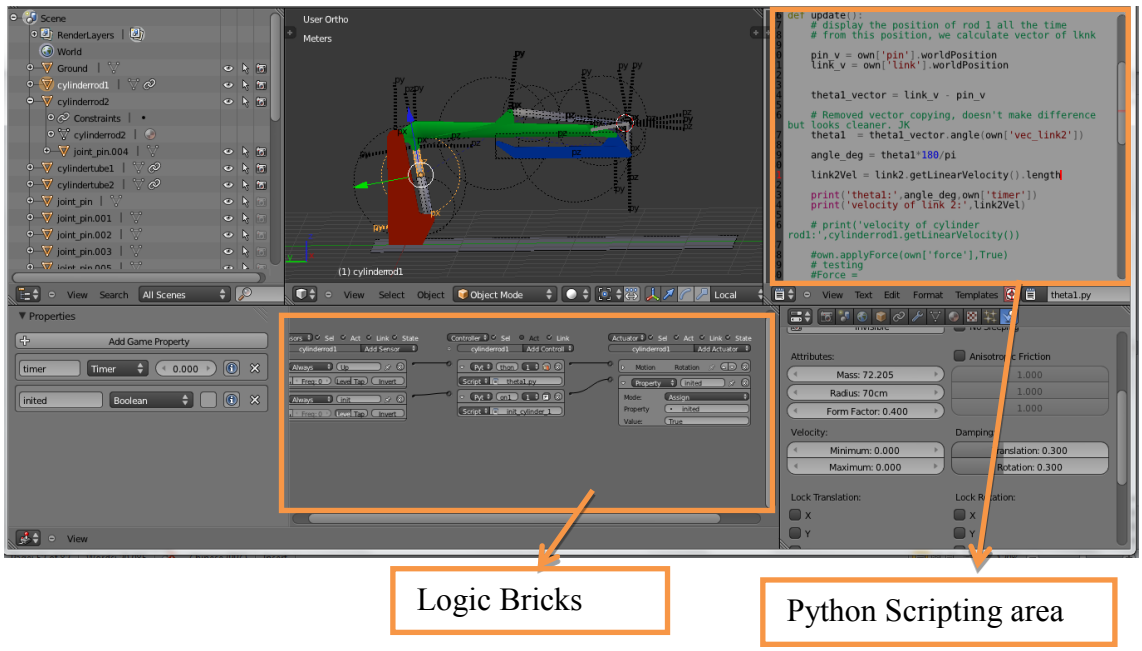


Figure 35: Logic Bricks and Python scripting for Crane simulation

3.2.3.1 Game Logic Bricks

Blender game engine provides a visual programming tool, called game logics, where interactive events or behaviors can be developed intuitively and it allows game users to develop the event trigger mechanism of a game using drag and drop controls. (J R Juang,W H Hung,S C Kang, 2010) The relationship between user inputs and system responses by this panel as defined in the figure 34.

There are three major components in logic bricks; sensors, controllers and actuators

respectively. During one simulation, three logic bricks work in a sequence, such like sensors sense if events as any inputs take place, such as a collision, a key press or mouse movement. Sensors are linked to controllers, which compares arguments in Python or any condition listing in the controllers and thus activate or deactivate actuators. (Blender Game Engine Overview) To be more specific, sensors are the causes of logic to do anything. This can be a trigger event such as a nearby object, a key pressed on the keyboard, timed events, etc. When a sensor is triggered, a positive pulse is sent to all controllers that are linked to it. The controllers are the bricks that collect data sent by the sensors. Actuators perform actions, such as move, create objects, and play a sound. The actuators initiate their functions when they get a positive pulse from one (or more) of their controllers. (Blender Game Engine manual, 2009)

3.2.3.2 Establishment of logic brick for manipulating crane

Generally, all logic bricks are created to exert external forces to simulate pressure inputs in cylinder chambers, which motivates crane joints to rotate based on the physics constraints which have been made in game physics and then game engine will calculate the rotation angles of crane model.

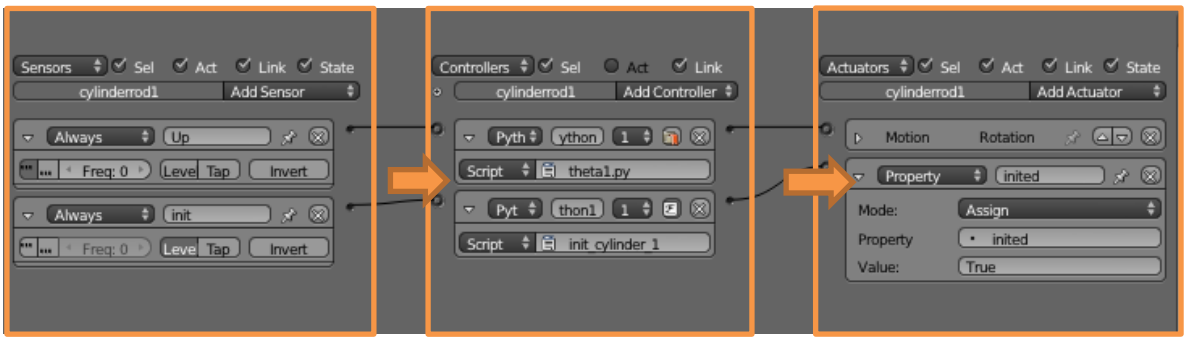


Figure 36: Sensors, controllers and actuators of the Logic context.

Logic bricks for controlling crane joint θ_1 have been expanded so that each sub-panel can be viewed individually. There are two sensors, two controllers and two actuators in control logic bricks. The Always sensors at leftmost are working as positive pulses' senders to activate different actuators. Two Python script controllers are designed to declare and initialize the objects (init_cylinder_1.py) and another for applying external forces (theta1.py) as well as calculate two rotational angles via vectors.

In order to calculate joint angles correctly, the initialization part has to be done as first step to declare initial positions of related crane components. During the entire simulation, as soon as initialization process is done, python controller's collects data obtained from Actuator 'Motion' shown in the figure below. This actuator is responsible for applying external forces along cylinders' local Z axis as the general crane's input. As crane starts moving, game engine is calculating and updating latest data about rotational joints based on following scripts. A newer value of joint angles will be printed

out each time Python controllers sends commands every key frame in game logic.

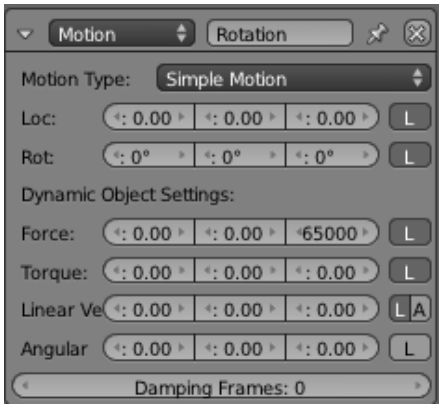


Figure 37: Simple Motion actuator used for exerting input forces.

3.2.3.3 Python scripts as a controller

As mentioned above, one Python controller is employed to control the 'Motion' actuator and thus to compute rotation of crane joints as shown in the figure 36. For example, the rotation angle θ_1 θ_2 can be easily calculated as the angle of related vectors, thus this assumption accelerates simulation process as Python has its own easy method to calculate angles between vectors. The main purpose of Python is to track and report locations of 2 links at each frame as well as to active Simple Motion actuator. Below listed the Python scripts to calculated rotation angle θ_1 as an example and each part of scripts are inserted with explanations in detail.

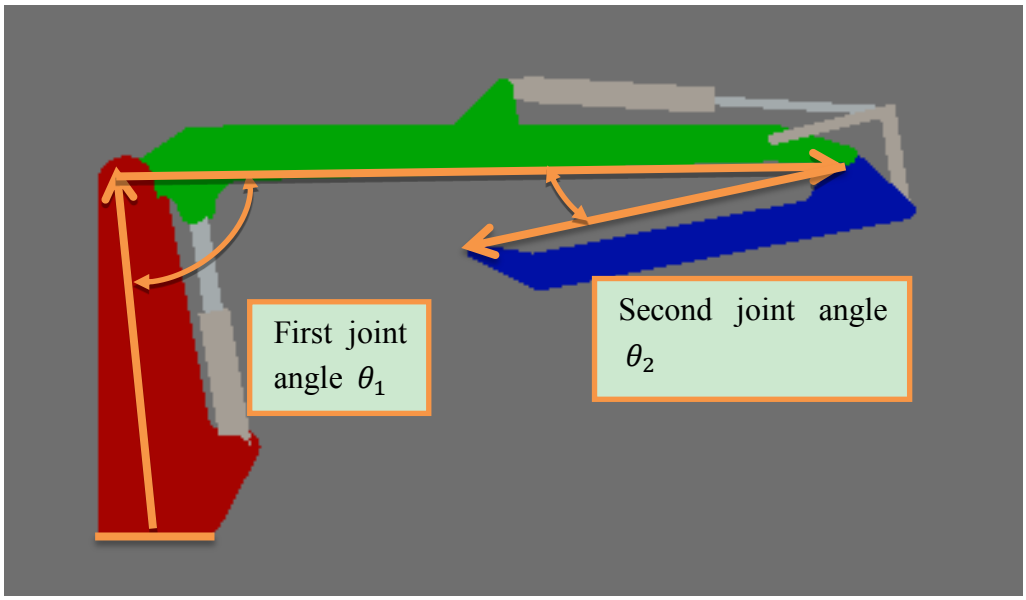


Figure 38: Vectors and angles' clarification in Blender

The first part of this script is to import the game engine and mathematic libraries and

then declaring controllers, several actuators and components of crane from current active scene. By defining the location of those items, it is possible to calculate rotation angles with the help of locating vectors of closer objects. In addition, the scripts also help to check if the inertia tensors of objects are correct or not.

```
import bge
import mathutils
from math import pi

output_file_name = "theta_1.txt"
def init():

    cont = bge.logic.getCurrentController()
    sensor = cont.owner.sensors["Up"]
    own = sensor.owner
    scene = bge.logic.getCurrentScene()
    link2 = scene.objects["link2"]
    link3 = scene.objects["link3"]
    pin  = scene.objects["joint_pin.002"]
    cylinderrod1 = scene.objects['cylinderrod1']
    cylinderrod2 = scene.objects['cylinderrod2']

    print("Local inertia of link2 : ","link2.localInertia)
    print("Local inertia of link3 : ","link3.localInertia)
    print('Local inertia of cylinderrod1 :', cylinderrod1.localInertia)
    print('Local inertia of cylinderrod2 :', cylinderrod2.localInertia)

    own['link'] = link2
    own['pin'] = pin
```

The *init()* function is used to obtain initial positions of cylinders, link 2 and base that have been pre-defined in the *main()* function. The values that has assigned cannot be changed during process because those items calculate the initial value of θ_1 as reference.

```
    # This part is used to initalise all the position of links
    own['init_rod1'] = mathutils.Vector((-0.000146,-0.29,1.239))
    own['init_link2'] = mathutils.Vector((0.000441928,-1.6,1.9634))
    own['init_joint1'] = mathutils.Vector((0,0.15,1.90)
    own['vec_link2'] = own['init_link2'] - own['init_joint1']

    # Output file
```

```
        # own['file'] = open(output_file_name, 'w')
        own['force'] = Force
        # Set the inited flag to true
        cont.activate('inited')

    print('In init cylinder 1 script')
    init()
```

This scripts ends with a print and it sets the property 'inited' to be true so that following update scripts can run frame by frame. The *update()* function shown below updates world position of link 2 as soon as game engine starts the simulation. Thus, new rotation angle will be obtained if the world position of link 2 gets updated each frame approximately 60 frames per second. Onwards, rotation angle can be easily calculated based on vectors of link 2 and base via Python method.

```
def update():
    # display the position of rod 1 all the time
    # from this position, we calculate vector of link

    pin_v = own['pin'].worldPosition
    link_v = own['link'].worldPosition

    theta1_vector = link_v - pin_v

    # Removed vector copying, doesn't make difference but looks cleaner. JK
    theta1  = theta1_vector.angle(own['vec_link2'])
    angle_deg = theta1*180/pi
    link2Vel = link2.getLinearVelocity().length

    print('theta1:',angle_deg,own['timer'])
    print('velocity of link 2:',link2Vel)

    act = cont.actuators['Rotation']
    cont.activate('Rotation')

# We need to wait till the init script has been ran
if own['inited'] :
    update()
```

Finally, the rotation angles as well as inertia and velocity of links and cylinders will be displayed in toggle console one by one at each frame.

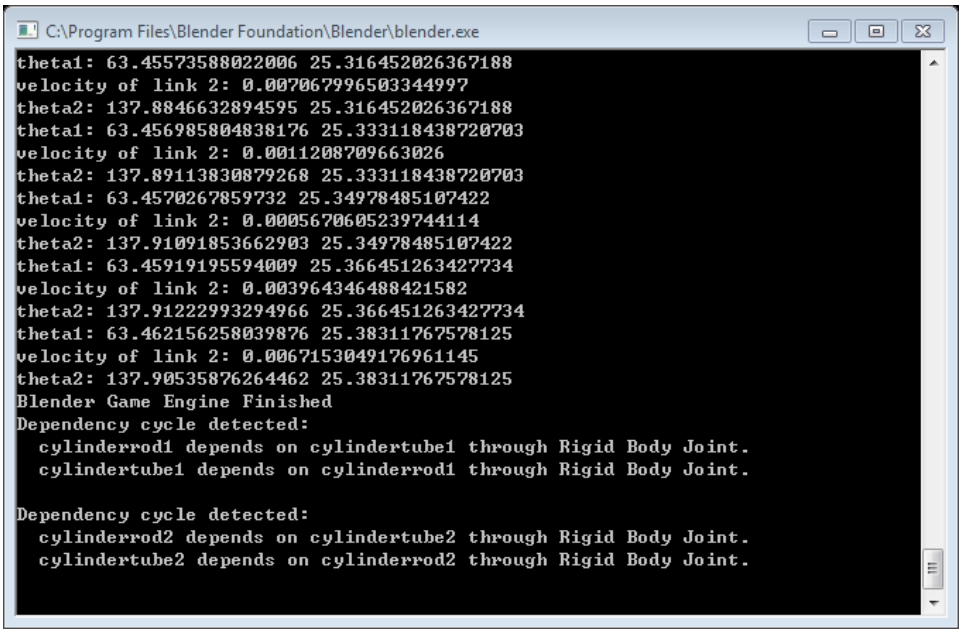


Figure 38: representation of result displayed in Blender toggle console

Joint angles are calculated according to geometry analysis by Python and printed the result out in the toggle console with string format. The simulation of crane is performed based on game unit key frames and activated by every single pulse input from logic sensors. The converter from key frame of Blender to time unit in real life is 1/60 frame/seconds. In this case, the entire simulation runs sixty times per seconds in Blender.

3.3 Implementation of simple simulation in game physics

So far, users have accomplished three major steps to make preparation for simulation of crane model, which are steps of having set crane parameters, game physics features and created logic bricks according to specific requirements. A series of pictures are caught for showing whole simulation process using a game engine. In this environment, physics effects are built to support realistic dynamics motion. Thus, all the objects make collision responses and are influenced by the gravity and dynamic forces. (J R Juang,W H Hung,S C Kang, 2010)

3.3.1 Development environment of the virtual crane and pendulum

The virtual simulation is implemented in Blender 2.61 version, including a built-in game engine and a bullet physics application. The physics engine is based on Bullet Physics, a free physics library for building 3D games.

The hardware environment for developing simulation included a laptop: one HP G62 with AMD P520 Dual-Core Processor 2.30 GHz and Vision AMD graphics card.

3.3.2 Scenario design for crane

The crane model performs one activity that there is one object located at a higher position, crane is operated to rise in order to reach this particular object as a simulation of lifting woods in real life. It is a complicated example of crane motion in game physics when taking every aspect of physics and gravitational environment into account.

The input of this scene is two hydraulic pressures obtained from two cylinders. The cylinder forces inputs of the whole system are provided as below.

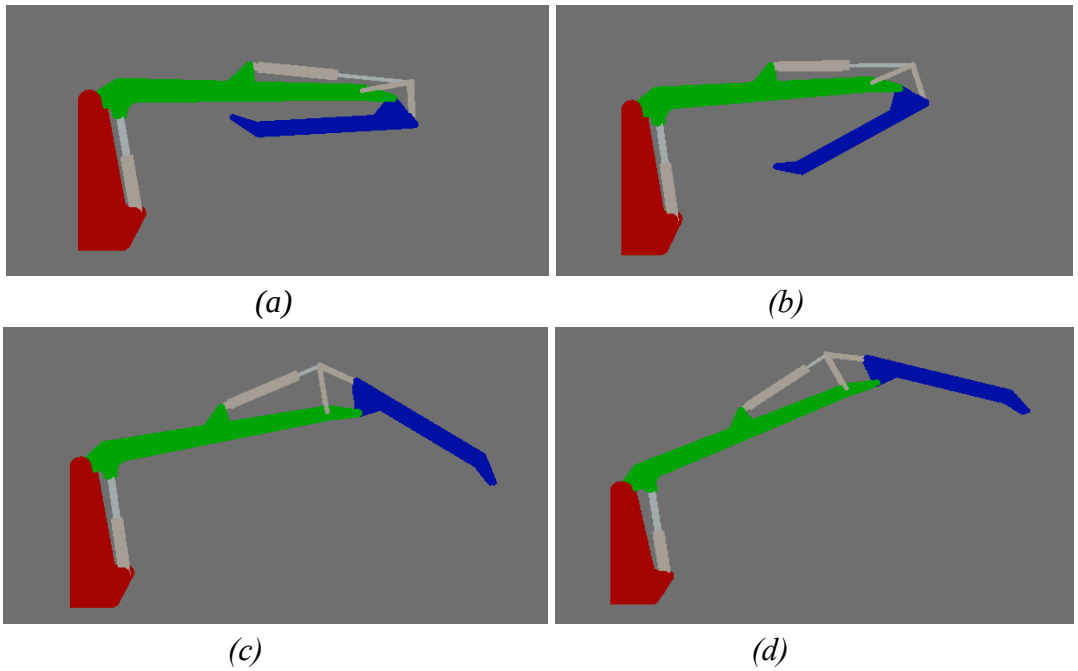
Corresponding cylinder 1 force: $f_1 =$ 65000N

Corresponding cylinder 2 force: $f_1 =$ -7000N

3.3.3 Crane simulation in game physics

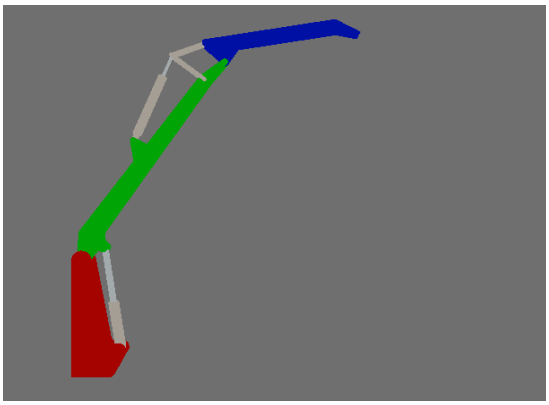
The entire simulation takes places within 10 seconds till crane gets in stable position. The result below shows a smooth and stable simulation without any potential crashes of the crane. Users can be immediately involved in the simulation and easily understand the process of the lifting operation by manipulating and playing with the virtual machine in this environment.

As simulation starts, link 2 is growing rapidly from its initial position because corresponding cylinder retracts to the minimum position in a short time. Meanwhile, link 1 moves slightly from the very beginning and it extends to the maximum position that has been achieved. The whole implementation is shown in the figure below.



(a) *(b)*

(c) *(d)*



(e)

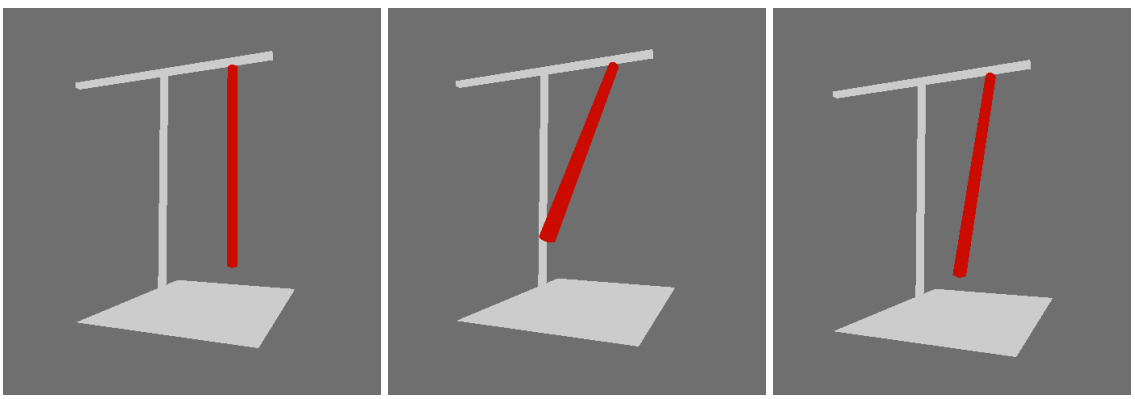
Figure 39: (a) Crane is at its initial position ;(b) Cylinder starts the simulation;(c) Cylinder 2 starts to retract; (d) Cylinder 1 starts to extend; (e) Cylinder2 extends to maximum while Cylinder 1 stays at its minimum place.

After attained position that crane model is able to reach, joint angles will be in to balance condition and stay at the equilibrium point stably. Judging from clips shown above, the simulation result shows that Blender game engine is stable without any crashes in emulating crane behaviors in general.

3.3.4 Pendulum simulation in game engine

Unlike complex crane simulation, the pendulum represents a simple simulation in Blender game engine. The pendulum does not include any complicated mechanism and dynaimc cylinders so that it can be regarded as a single rigid body that has no effect on other componenets. Therefore, it present the game physics clearly without any disturbance other than the external forces.

The boom just is exerting on two forces, gravational forces of 20N and external forces of 60 N that starts the pendulum wondering off shortly. The frequency that pendulum going for one round makes another reference on the game physics.
System input: external forces 60N applying at center of the hanging boom.



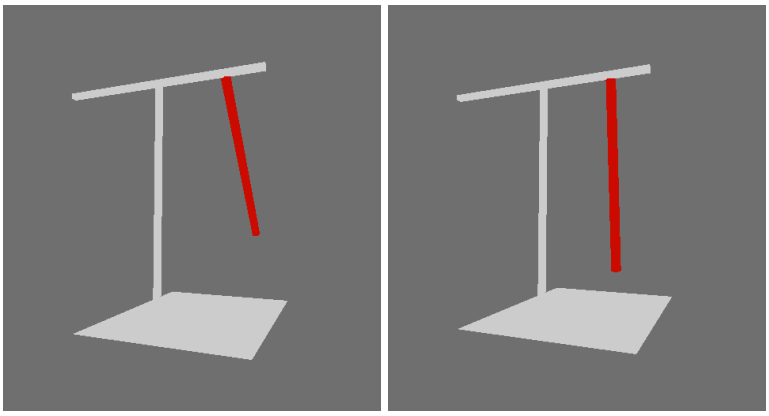


Figure 40: The pendulum to finish one round in game engine

The overview angle that pendulum has passed through is displayed in the figure 41.The maximum value of the pendulum is 18 degree and it reduces the angle every single round till it stops.

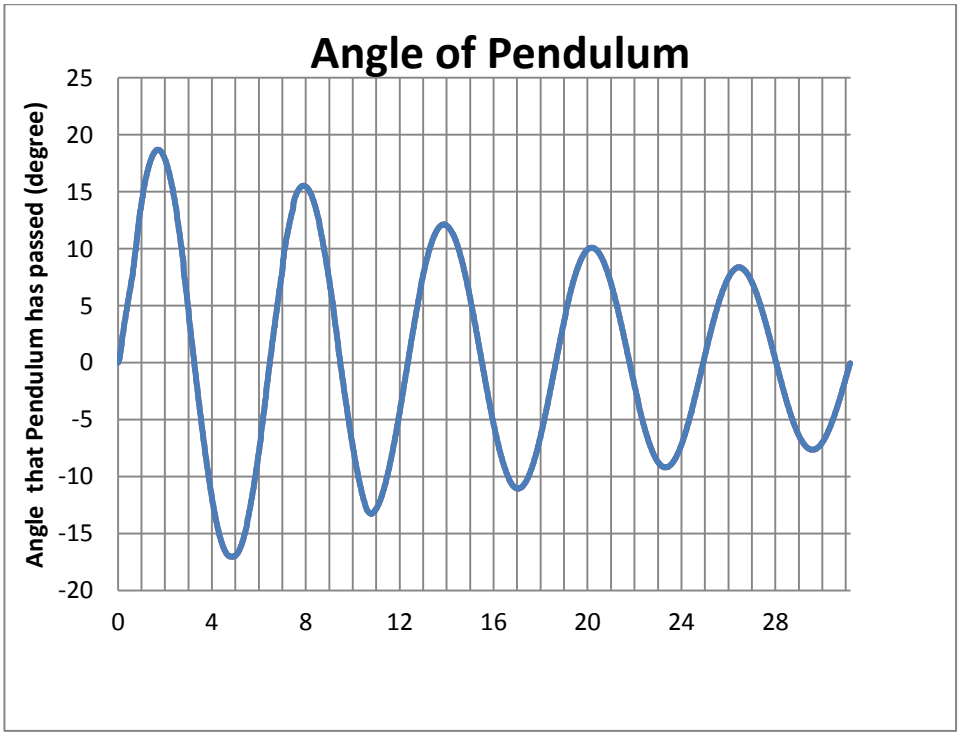


Figure 41: The frequency that pendulum simulated in Blender

4 Comparison of BGE and Matlab approaches

During this section, the comparison demonstrates on whether the performance of game

physics simulation is reliable and stable enough to be used to optimize the development of mobile machine with virtual reality technology. In particular, if game physics can be utilized in giving necessary trainings for fresh machine operators to handle heavy-duty machines before start working as well as to ensure safety of mobile machine as early as possible.

In order to identify reliability of game physics, a crane model is being tested with different pressure inputs obtained from crane cylinders, multiple inputs will lead to different levels of complexity in system variables for performing the virtual crane. All data of outputs that have been collected in simulation will be valued according to two aspects, which are angular displacement of joint angle, rise to be stabilized, and thus individual cylinder movement will be investigated.

4.1 Objective of comparison

There are two alternatives to test accuracy of Blender game physics, angular displacement performance of crane joints and time to reach steady state. Among these aspects, the displacement performance of crane will be regarded as first reference in evaluation than other requirements. According to the equation (2), any position in crane's reachable range is directly proportion to the rotational displacement of each joint in terms of forward kinematics. That is to say, no matter how much forces crane obtained, those reachable positions are unchangeable. Therefore, it is an explicit reflection of game physics without considering the force that causes the motions.

Evaluation of accuracy of game physics is implemented by calculating the resulting rotation angles of crane joints qualitatively based on the game engine and Matlab approaches.

4.2 Identicalness of the model

Since the crane model is being created and evaluated into two different methods, from computation tool in Matlab to graphics tool Blender, keeping model as identical as possible is of great importance under the topic. In general modeling, forward kinematics is employed in Matlab to link crane components to multi-dynamics while Blender uses a similar kinematic chain to link all components. However, it is nearly impossible to make both crane models exactly the same in the general modeling process due to different methods.

For example, the inertia tensor of second link in Matlab is computed as [485.33, 5.82, 481.9], while the one in Blender is shown as [472.80, 462.54, 15.73] due to different ways to define inertia tensor.

4.3 Comparisons on 2DOF joints of crane

Each case in comparisons part is composed of two evaluations of Game physics, the

angular displacement of joints and rise time to get maximum displacement. In this research, angular displacement performance of crane model denotes the angular rotation of crane joints around local Z axis, counting from the initial position to the peak value that crane has achieved. The initial angles are 0° for both joints, therefore, the angular displacement is the gross rotation that crane links have been passed through.

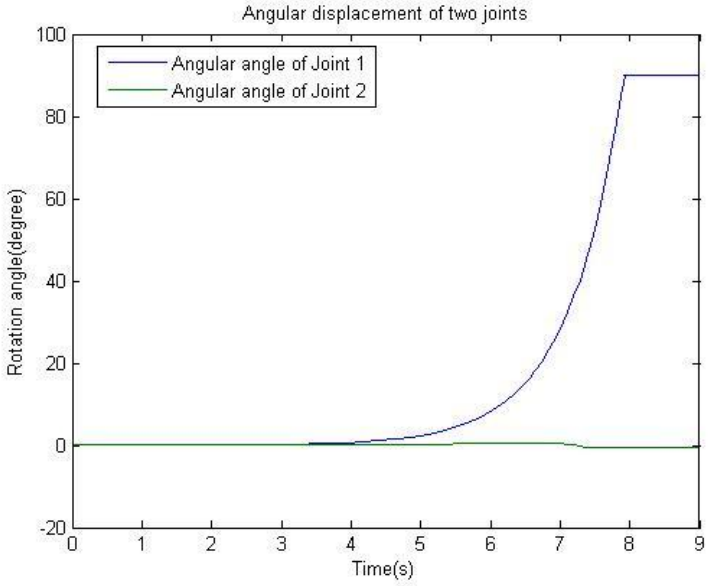
The angular displacement capacity is the main factor to judge accuracy of game physics among those aspects. As explained on previous session, position of each component of crane model is partly dependent on rotational angles based on forward kinematics. Hence, any unrealistic performance in game physics angular displacement could be easily sensed by angular displacement.

Time to get stabilized in steady state is another evaluation to access accuracy of game physics. Time to reach steady state denotes the gross time consumption elapsed from the moment that game engine starts its execution till when output reaches the steady state. Generally speaking, the time for crane to get stabilized defines the extent to how fast could crane react to external forces in BGE and Matlab approaches.

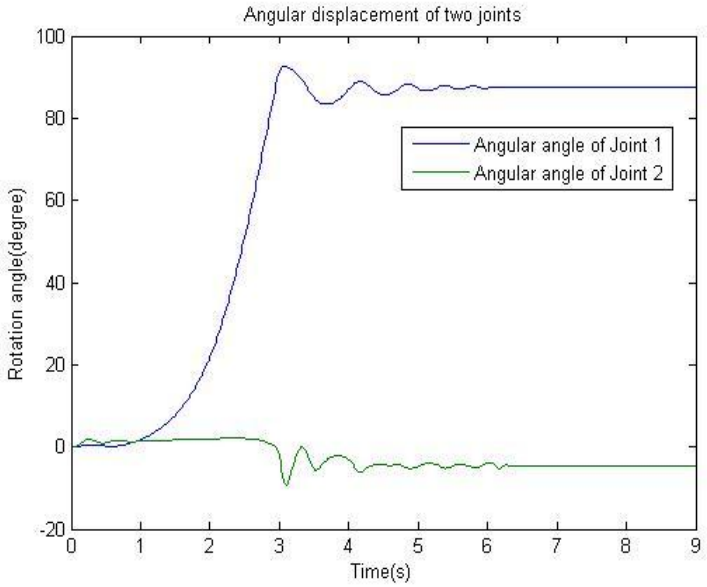
4.3.1 Equilibrium force to keep in its initial position

The equilibrium point denotes the moment that entire crane gets balanced so that its links keep in a static condition. Therefore, the angular velocities and accelerations of components reduce to zero. According to equation (47), the corresponding equilibrium forces are $f_1 = 138430$ and $f_2 = 36347$ for both cylinders.

After taking those forces as system inputs, the angular displacement and time performance can be displayed in the following plots in both cases:



(a) Angular displacement of 2 joints in Matlab



(b) Angular displacement of 2 joints in Blender

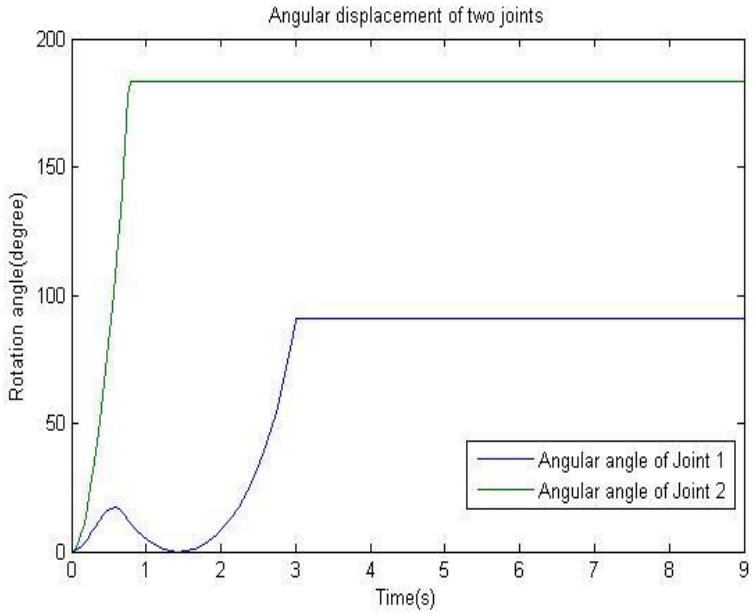
Figure 39: Rotation angles of joints via BGE and Matlab approaches when cylinder pressure inputs are $f_1 = 138430N \quad f_2 = 36347N$

As shown in the figure 39 above, steady value of first rotation angle is achieved at around 90 degree for both cases. Meanwhile, it takes approximately eight seconds for crane to settle down while it costs three seconds for Blender crane model to get to maximum position and another three seconds handle the hard damping happening at the limits of cylinders.

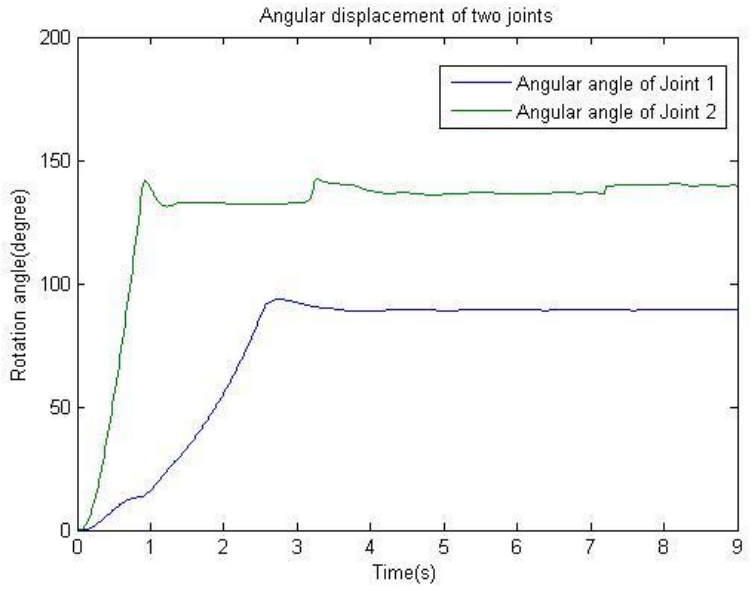
For the second joints, despite the oscillation caused by hard damping from first joint in Blender, it stays at its initial position all the time at equilibrium point.

4.3.2 Forces to lift crane to its maximum position

In this section, any random forces that can lift crane from its initial forces obtained from equation (46), are used to testing the accuracy of game physics by evaluating final position of joints. In this case, the force is chosen to be $f_1 = 70000N$ and $f_2 = -7000N$. The angular displacement and time performance are plots as follows:



(a) Angular displacement of 2 joints in Matlab crane



(b) Angular displacement of 2 joints in Blender crane

Figure 40: Rotation angles of joints via BGE and Matlab approaches when cylinder pressure inputs are $f_1 = 70000N \quad f_2 = -7000N$.

The plots regarding input forces $f_1 = 70000N \quad f_2 = -7000N$ are plotted to make comparison. It can be seen that first joint of crane takes three seconds to settle down to steady state while second joint takes around one second to reach its maximum position in both cases.

As for two rotational angles in Matlab, the second joint has a negative effect for the first joint to reach its maximum position. When the second joint gradually approaches to 180 degree, it gives the second link one force that is on the opposite direction of its movement, which results in reducing angular displacement for the first joint. So does the Blender crane. Therefore, the second joint can be regarded as a disturbance input for

the first joint.

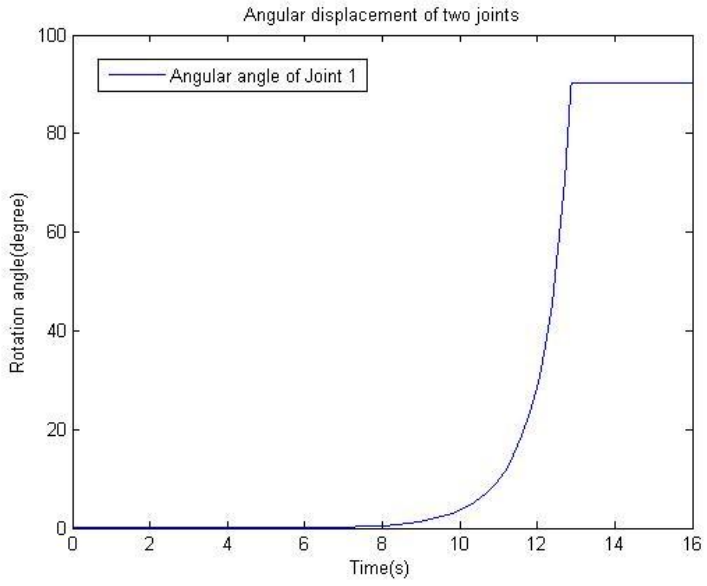
Based on the existing crane simulation on game physics, the result shown in game physics is similar to the Matlab result in terms of time to reach steady state, angular displacement of joint angles for those cases. Therefore, game physics simulation is accurate to simulate complex mobile machine in general.

However, since the second cylinder along with torque mechanism makes too much uncertainty on the first joint, it can be seen as a disturbance input of system. Therefore, the second cylinder, torque mechanism and third link will be excluded to improve the accuracy of game physics by simplifying the model itself. Thus, next section will investigate the simpler crane model with one cylinder to further evaluate the performance of game physics.

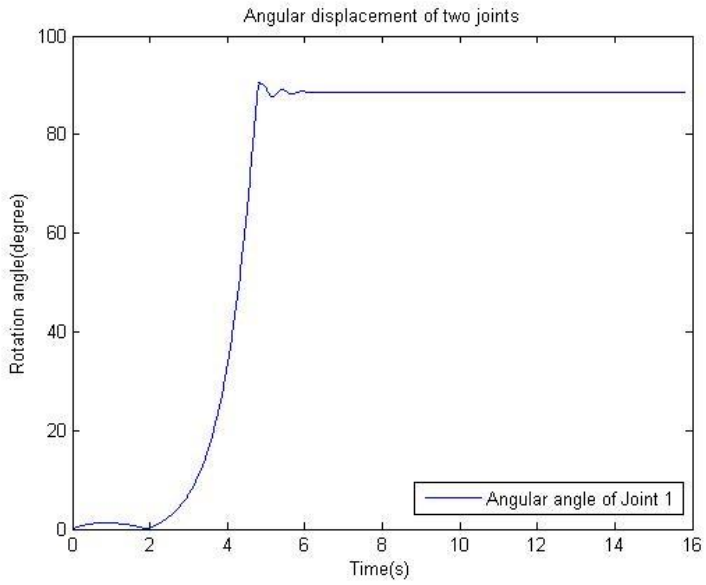
4.4 Comparisons on 1DOF joint of crane

4.4.1 Equilibrium point to keep in its initial status

A simpler model without second cylinder and torque mechanism is employed to minimize any disturbance obtained from second cylinder. Based on the equation (44) and (45), the equilibrium point of simpler crane, first cylinder and second link can be calculated with same fashion as previous equilibrium point. In this case, the only cylinder force $f_1 = 28783N$ in Matlab. The plots below display the angular displacement of simpler crane when taking same cylinder force as system inputs.



(a) Angular displacement of first joint in Matlab crane



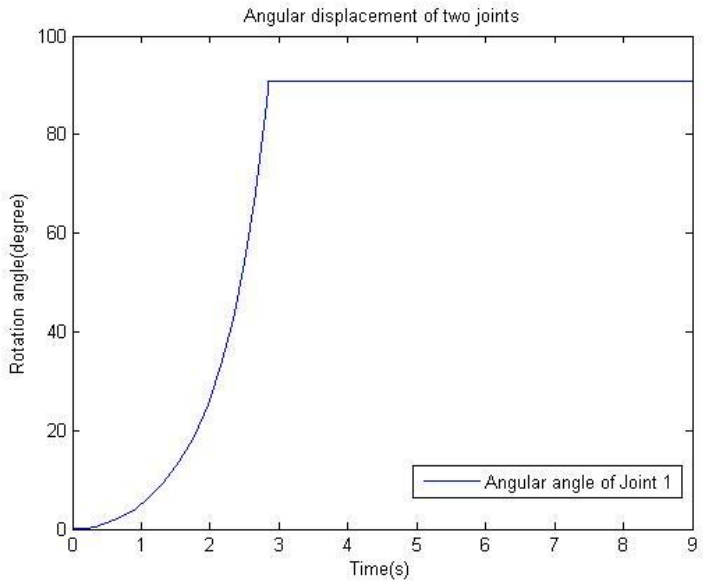
(b) Angular displacement of first joint in Blender crane

Figure 41: Rotation angles of joints via Matlab and Blender approaches when cylinder pressure inputs are $f_1 = 28783N$.

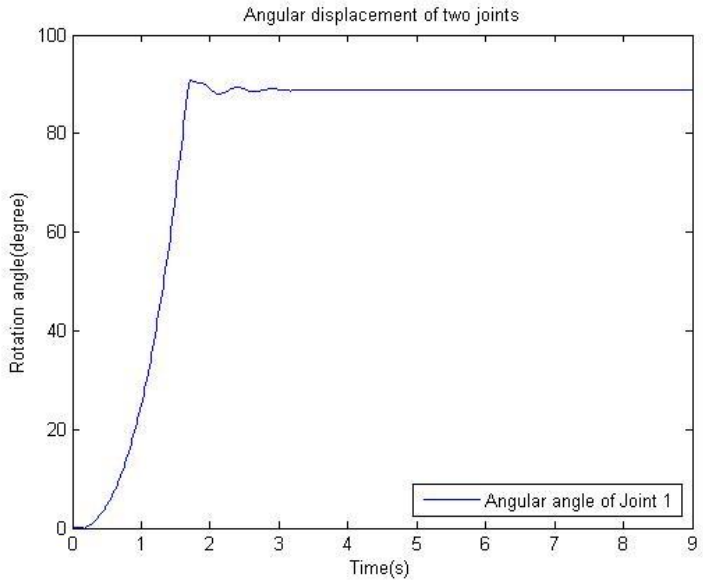
As shown in the figure 40 above, steady value of first rotation angle is achieved at around 90 degree for both cases. However, Matlab crane stays at in its initial position for 8 seconds and 13 seconds f to settle down, while Blender crane stays at its initial position for 2 seconds and costs 6 seconds to steady state. Moreover, first joint in simper model has less oscillation compared to its original model.

4.4.2 Forces to lift crane to its maximum position

In this section, any random forces that can lift crane from its initial forces obtained from equation (46), are used to testing the accuracy of game physics by evaluating final position of joints. In this case, the cylinder force is $f_1 = 30000N$



(a) Angular displacement of first joint in Matlab crane



(b) Angular displacement of first joint in Blender crane

The plots regarding input forces $f_1 = 30000N$ are plotted to make comparison in between Matlab and Blender crane model. In both case, the first joint get to its maximum position of 90 degree in a short time. Moreover, it can be seen that first joint of Matlab crane takes three seconds to settle down to steady state while Blender crane takes two seconds to reach its maximum position and another two seconds to settle down.

4.5 Result of performance comparisons

4.5.1 Conclusion on comparisions analysis

Generally, by developing an interactive physics-based simulation of crane model using game physics, several pros and cons of game engine method has been drawn in comparison with a Matlab methodology. As a summary, the result of this research demonstrates the development of simulation delivering high quality physics is efficient and stable. A more detail result in previous discussion chapter will be discussed and thus, comments and solutions about improvement of Blender game physics are provided.

4.5.1.1 Comparison of 2DOF crane

The first evaluation shown in table below is with system inputs: $f_1 = 138430N$ and $f_2 = 36347N$, the angular displacement and time performance are shown

Table 1: Conclusions of angular and time performance of crane in equilibrium point.

	Angular displacment performance(degree)		Time performance(second)	
Method	**Joint 1**	**Joint 2**	**Joint 1**	**Joint 2**
Matlab	90.15	-0.49	7.90	6.58
Blender	87.52	-4.62	7.11	4.60

If the cylinder forces changes to values $f_1 = 70000N$ and $f_2 = -7000N$, that can lift crane to its maximum position, the angular displacement and time performance are shown as follows:

Table 2: Conclusions of angular and time performance of crane when forces are values $f_1 = 70000N$ *and* $f_2 = -7000N$.

	Angular displacment performance(degree)		Time performance(second)	
Method	**Joint 1**	**Joint 2**	**Joint 1**	**Joint 2**
Matlab	90.60	183.50	3.00	0.76
Blender	89.20	139.20	3.86	0.96

4.5.1.2 Comparison of 1DOF crane

The simpler crane excludes the second cylinder and torque mechanism, which can be regarded as disturbance input since second cylinder influence displacement performance of first joint in a negative way. The first evaluation shown in the table below is a 1DOF crane with system inputs: $f_1 = 28783N$

Table 3: Conclusions of angular and time performance of crane when force is $f_1 = 28783N$ *as its equilibrium point.*

Method	Angular displacement performance(degree): Joint 1	Time performance(seconds): Joint 1
Matlab	90.15	12.86
Blender	88.58	6.607

If the cylinder force changes to the any values, $f_1 = 30000N$, that is able to lift the crane to its maximum position, the simpler model with one cylinder can be plotted regarding angular displacement and time performance as:

Table 4: Conclusions of angular and time performance of crane when force is $f_1 = 30000N$

Method	Angular displacement performance(degree): Joint 1	Time performance(seconds): Joint 1
Matlab	90.69	2.89
Blender	88.91	3.01

If the value changes to $f_1 = 40000N$, the angular displacement and time performance are shown as follows:

Table 5: Conclusions of angular and time performance of crane when force is $f_1 = 40000N$

Method	Angular displacement performance(degree): Joint 1	Time performance(seconds): Joint 1
Matlab	90.39	1.30
Blender	89.38	1.15

All data listed in the comparison parts are crane's performance on angular displacement of joints and time consumption to get to steady state in game physics with respect to ones in Matlab approach. Based on the summary of graphics for 2DOF crane, the Matlab and Blender video has certain difference as there is one disturbance input, the force from second cylinder. However, after considering a simpler crane model, the angular displacement measured in Matlab is similar to the result obtained from Blender. On one hand, basic simulation in game physics is realistic and believable according to the overview of data. On the other hand, game physics in Blender is able to handle simpler crane with 1DOF better than original 2DOF crane.

As a conclusion, even though the results collected from game physics are not

accurate to render such a complicated 2DOF crane, the result regarding 1DOF is more reliable to Matlab simulation. Therefore,

4.5.2 General evaluation of two mythologies

Based on the two methodologies that have been analyzed, benefits of using game physics and traditional programming to develop simulation are summarized in the following table. The conclusion show openness to game physics compared with traditional programming method from a general view.

Table 6: The pros and cons of developing physics-based simulation using game engine and traditional Matlab method. (J R Juang,W H Hung,S C Kang, 2010)

Compared Requirements	Employing Game Engine	Employing Matlab method
Interactive feedback of system	High	Very Low
Accurancy of physics simulation	Limited	Dependent
Required programming skills	Low	High
Focus on developing simulation method	Low	High
Extendibility and resuabililty	Limited	Dependent
Prototyping	High	Low

Regarding the interactive feedback from system, the game engine provides users with a quality interactive feedback of system response. Whereas, utilizing Matlab may lead to delay because there are no such instant interaction or information about mistakes are provided in the Matlab. Therefore, the interaction with programmed environment in Matlab is indirect and invisible.

Exploring a better and efficient environment to develop real-time interaction is a critical task for existing game physics engine. The accuracy of physical simulation partly depends on physics scripts that are created to duplicate realistic behaviors obtained from physics laws, partly on the complexity of object that game engine is dealing with.

Thus, game engine is a big calculator, it makes easier to model and simulate objects. However, users need to follow the way how game engine calculates. Therefore, the core computation of game physics in Blender is hard to access and modify as it is integrated into complicated designers' programming. Unlike game physics, Matlab is more focusing on development of simulation method.

Employment of Matlab is more accessible because the environment is decided by software developers not by fixed function panels like in Blender. In other words, designers are able to develop physics environment with their own ways. In sum, transitional programming might be the only approach to create an advanced graphical effects and additional physics behaviors such as mass, collision response and physics

effects. On the other hand, an accurate environment needs correct details in every step and thus takes time to develop the whole procedure.

For the extending virtual applications such as Blender, 3D max, game engine is easier to extend and has new add-on applications or features compared to additional programming. However, it is still limited in the development because game physics is hard to integrate with others application in different areas. On the other hand, Matlab is open for different purposes in various fields and thus Matlab programming packages to develop simulations is allow a high degree of extendibility and reusability with numerous add on features, such as Robotics toolbox for Robotics. (J R Juang,W H Hung,S C Kang, 2010).

4.6 Potential challenges in game physics

Even though game physics technology has evolved recently to reach the most outstanding performance with powerful computer graphics to facilitate industrial usage. However, few uncertainties still exist related to game physics because it is too simple. This technology is yet far from being persuasive and takes time to improve to be professional such as generation of inertia tensor, defective feature in collision detection, immature physics constraints. Potential problems of game physics are explored and explained in detail.

4.6.1 Lack of inertia sensor of objects in game physics

With comparison to great inertia tensors that could be pre-set in Matlab scripts, it is no easy way to apply any pre-defined inertia tensors in Blender game physics because inertia tensors have been generated automatically in Python as soon as objects are created and thus the codes for matrix inertia tensors are not accessible for users.

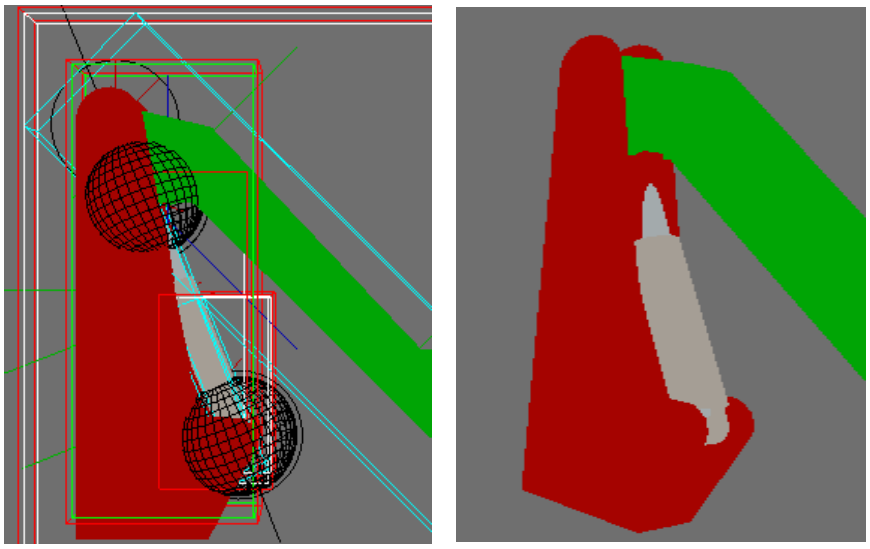
As the inertia tensor is associated with the rotational speed of object. Therefore, by reducing the inertia tensor, the rotational rate is increasing. Consequently, it is probably one of the reasons that result in difference in angular displacement via two methods.

4.6.2 Immature physics collision of objects

Physics constraints and collision are the most useful features in game physics because those help users to set up object's physical relationship with other component and set displacement limitation for each item. In physics constraints, displacements limitations are set to avoid any collision occurring in between the object and its neighboring objects. On the other hand, if displacement limitation of objects is defined incorrectly, the collision detection feature should be capable of detecting and stopping any colliding movement.

Collision detection is achieved by detecting if there are any overlapping spots based on the geometry arrays of objects. However, if the overlapping spots come with more

than two overlapping spots, it is possible that there will be missing collision spots left, which will result in undetectable collision between objects. For example, collision occurs at the crane base in red when the first link in green is going down in the following figure. It could be caused by too many collision bounds overlapping in the same location for different components as shown in the figure 44(a). When too many bounds are mess up in one limited position, it becomes harder and time-consuming for game engine to detect any different collision there. The consequence of mixed collision is shown in figure 44(b).



(a) (b)

Figure 44: Collision occurs due to too many overlapping collision bounds with physics visualization.

4.6.3 New spring constraint in object contraints

For the current game engine, it is not possible to model the spring behavior with game physics.

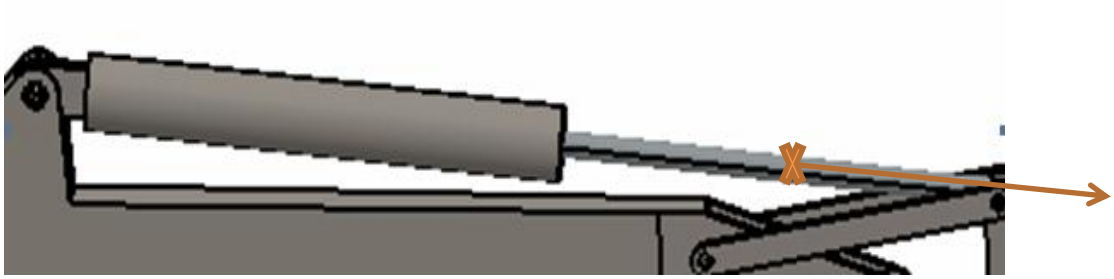
Compared to other game developing software such like 3D Maya, a spring constraints could be added to rigid body joint, so that it makes rigid bodies move as if they are connected together by a spring-like shaft. They provide one degree of freedom in translation based on the spring's elasticity, but complete freedom in terms of rotation. The spring constraint is based on Hooke's law which states that the force a spring exerts is directly proportional to how much it extends (its length). (Spring Constraints)

The bodies attached to springs can move around freely relative to each other. The effect is often similar to objects tied with an elastic cord, with the rigid bodies stretching and bouncing back. It is also possible to offset the position of the spring so that it's not directly connected to the rigid body's center.

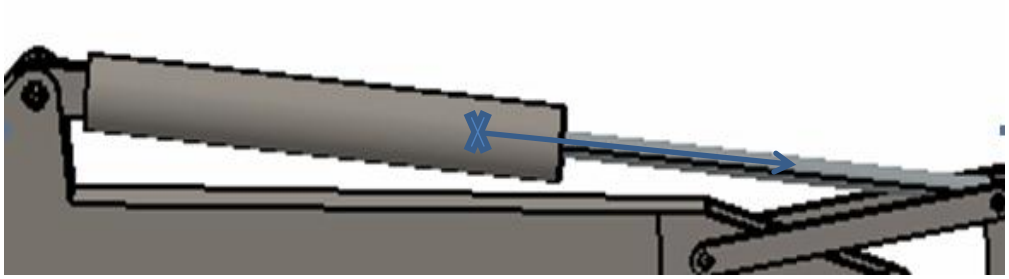
Moreover, it also can used to simulate other complex mechanical components that exist in the vehicles such as shock absorber in motorcycles or any elements with good elasticity.

4.6.4 Limit location of function object in Python dynamic

In game physics, there are two alternatives of Python dynamic inputs to activate the cylinder force in Python, which are *applyForce(force, local)* and function *applyTorque(torque, local)*. Where *force* and *torque* denote the quantities of force or torque that apply to the crane cylinders, *local* decide if function inputs are generated based on local axis or world axis.



(a) Central position where cylinder pressure p_2 acts on in BGE



(b) Start position where cylinder pressure p_2 should act on in real life

Figure 45: One optimization about Python that can make.

It can be seen from code *applyForce(force, local)*, *applyTorque(torque, local)* that there is no choice to change the location where force or torque applies but exerting forces in local or world coordinate.

For instance, as shown in the figure 44, the cylinder forces normally are analyzed and calculated with respect to start point of cylinder rods as shown in the figure 45(b), because hydraulic pressure makes interaction with cylinder. However, cylinder forces have to exert on central of rod in Blender game engine shown in the figure 45(a) because there is no possible way to select the position of function objects, which makes force calculation inaccurate.

Therefore, the optimized Python code for dynamic inputs could be *applyForce(force,position,local)*, *applyTorque(torque, position,local.* In this way, users could offset the position of dynamic inputs to a better exerting point than central position.

5 Discussion

In this research, the comparisons that have been developed indicate that game physics makes reliable simulation with respect to Matlab result. However, as the game physics cannot be accurate enough to make complicated crane model but this research show that it has the possibility for game physics to make crane simulation more specific and real-time with the development of this technique. Moreover, industrial practice could take advantages of these methods and tools benefit in many aspects such like design of mobile machines, giving training for fresh operators. A list of possible advantages, disadvantages and potential challenges as well for employing game physics compared to traditional programming can be summarized as follows.

a) Easy and fast to develop projects:
Only three major steps of crane simulation are needed to develop projects, which are loading game object, setting physics features and establishing logic brick. Those procedures are simple and can be developed without any programming skills. Users are able to construct objects easily with the help of useful physical functions and click-drag to connect the logic bricks to control objects. Therefore, if game developers intend to use physics effects in more than one game, then putting effort into creating a physics engine now pays off when users can simply import it into a new game other than creating thousands line of code for rebuilding another physics effect.

b) Quality and stable simulation of objects:
Game physics provides users with the ability to use and manipulate effects in an amazing way, which substantially increases the levels of immersion, where one could personally feel "inside" a real scene. For example, game engine is able to build a cloth simulator for capes and flags, a water simulator for floating boxes in real life. (Millington, 2007). In addition, thanks to the high level of quality in simulation, all kinds of objects can be visualized in the user interface and manipulated based on physical environment.

c) Real-time visibility of reactions:
Unlike the numerical and time-consuming programming in Matlab, Blender provides designers with visualized interaction that can be seen simultaneously. An immediate of software response makes it easier for users to shoot troubles and potential defects of objects to speed up the time consumption that is used for solving existing problems.

Although game physics has been evolved recently, but for a developing game engine, there are still many spaces to make improvements in the future. Since the accuracy of

Game physics is the core attention of this paper, several unexpected behaviors of crane in the simulation because the correction of those behaviors matures game physics. Generally, those are the main drawbacks of game physics are into investigation in different fields.

a) Game physics for suitable level design:

For current game physics, it is almost impossible to achieve rapid processing without reducing efforts on quality of simulation. Therefore, it is common to assume that certain assumptions on physics law are made in order to speed up the simulation and it may result in less accurate than detail establishment of physics model in Matlab.

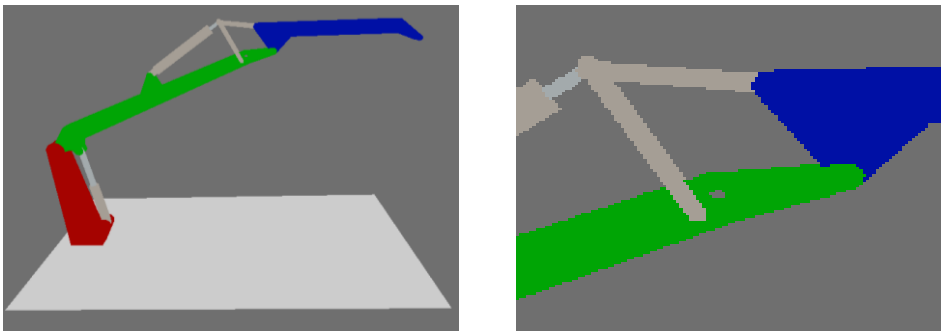
However, as quality and real-time simulation tends be the major focus of 3D application recently, how to keep game engine being both real-time and accuracy becomes important.

On the other hand, a general-purpose physics engine is quite processor intensive. Because it has to be general, it can make no assumptions about any certain kinds of objects it is simulating. When users are working with simple simulation, this generally can mean wasted processing power. However, if users are working with complicated simulation which takes time factor into consideration, the general physics can be rather simplified than the level that users need. For example, for the crane cylinder, it is filled with hydraulic fluid and all cylinder operations are actuated by hydraulic pumps in the real crane operation. It is complicated to simulate hydraulic cylinders in game engine because there is no substitute for hydraulic fluids combined with rigid bodies. However, in this research, simulation of cylinder is simply achieved by Python dynamic force inputs.

Therefore, a suitable level-design game physics that is changeable according to what users require could be a better solution.

b) Weak in object constraints:

The hinge joint constraints are used to make connection between neighboring links and cylinders. In the figure 45, the joint connecting torque link with 2^{nd} link works like a screw to fix all joints at one position, however, one link is not strong enough to fix at its end position because the same link suffers large forces obtained from cylinder rod 2. It is caused by weak stiffness at joints position so that object is not able to fix physics connection at a proper place and thus link gets misalignment as illustrated in the figure below.



(a) (b)

Figure 46: (a) The joint connecting torque links with 2nd link gets misaligned to its origin position (b) the zoom-in pictures of clip (a).

c) Visualized collision detection:

For any kind of simulation in virtual reality, it is inevitable to have collision in between different components or objects during simulation. Therefore, it is better to find out any potential collision that could occur and solve these issues in the early phase. More concern could be put on how to visualize those collisions before the simulation starts. As game engine in Blender, the physics visualization is sort of complicated because there are too many collision bounds displayed during simulation or many collisions could be too small to be detectable by users. Thus, making collision detection visualized in Blender is of great importance. For instant, game engine could distinguish potential collisions by using different warnings before simulation, such as alerts and warning in red with indicator of collision location.

d) Limit location on Python dynamic object:

In game physics, there are two dynamic inputs in Python scripts to simulate cylinder hydraulic force, which are function *applyForce(force, local)* and function *applyTorque(torque, local=False)* (Game Types (bge.types)). Where *force* and *torque* denote the quantities of force or torque that apply to the crane cylinders, *local* decides if function inputs are generated on local axis or world axis. There is no choice to change the location where force or torque applies but exerting forces in local or world coordinate.

Therefore, two dynamics inputs could optimize to *applyForce(force, position, local)*, *applyTorque(torque, position, local)* so that the exterting point could offset to other better location than its central position.

6 Conclusion and prospection

This research provided readers a way to distinguish reliability of game physics by aiming at reducing efforts and time required on developing mobile machines as well as giving necessary trainings to heavy duty machine operators with small amount of budget.

Thanks to game engine and physics, it is possible to simulate virtual objects in real time with general gravitational environment. Unlike Matlab modeling, forward kinematics of crane is achieved by object constraints function with visualization in game engine, which makes it possible to connect crane links and cylinders without complex calculation and only simple modeling steps are needed to implement. On the other hand, game logic function in Blender helps users to control the motion of crane with a simple click-and-drag movement with mouse. Finally, comparison on physics simulation programmed via Matlab and Blender game engine investigate the physics capacity of game physics for real-time simulation compared to traditional programming methodology.

The conclusion based on the comparisons of two methodologies can be drawn that it is the existing game physics is able to render simpler crane with 1DOF, the more degree of freedom model have, the less reliable the game physics simulation are. However, as this technique matures with time, it is possible that complicated hydraulic crane can be simulated in real-time in the near future.

In the future, relating features about virtual physics, for example, collision detection could be more sensitive to multi-collision and object constraints could have more dynamic joints to simulate soft and rigid body dynamics. In this way, object constraints are not used only in game render animation, but also for real-time physical simulation and thus improved to level up accuracy of entire game physics.

In addition, it would be better if an easy extension of physics python function could be promoted since new Blender and game physics renews its version quite often. In this way, the automation industry and its associating business could benefit from the virtual simulation application. Quite a few risky working tasks could be tested in the early simulation phases, and thus helps to accelerate the security for employees and reduce the cost of a product. Later on, standard libraries f related to Robotics or Automotive simulation could be developed to enhance the integration for virtual game engine with real business.

Reference:

Wikipédia. (n.d.). Retrieved from http://en.wikipedia.org/wiki/Blender_(software)

Blender Game Engine manual. (2009). Retrieved from wiki Blender: http://wiki.blender.org/index.php/Doc:2.5/Manual/Game_Engine/Logic/Sensors

3D Theory - Collision Detection. (n.d.). Retrieved from http://www.euclideanspace.com/threed/animation/collisiondetect/index.htm

3D Theory- Collision Detection. (n.d.). Retrieved from http://www.euclideanspace.com/threed/animation/collisiondetect/index.htm

Blender for robotics. (n.d.). Retrieved from http://people.mech.kuleuven.be/~bruyninc/blender/roadmap.html

Blender Game Engine Overview. (n.d.). Retrieved from Blender: http://wiki.blender.org/index.php/Doc:Manual/Game_Engine

Bullet (software). (n.d.). Retrieved from Wikipedia: http://en.wikipedia.org/wiki/Bullet_(software)

Chi H L, Hung W.H,Kang S.C. (2007). *A physics-based simulation for crane manipulation and cooperation.* Pittsburgh,USA: Proceedings of computing in civil engineering conference.

Erleben K., Sporring J.,Henriksen K. and Dohlmaan H. (2005). *Physics-based animation.* Boston,USA: Charles River Media.

Forestry cranes. (n.d.). Retrieved from http://www.grues-forestieres.com/ENG/eng%20products.htm

Game Types (bge.types). (n.d.). Retrieved October 04, 2011, from Blender v2.59.3 - API documentation: http://www.blender.org/documentation/blender_python_api_2_59_3/bge.types.html

Grigore Burdea, P. C. (n.d.). *Virtual reality technology.* Wiley-IEEE.

Heinze, A. (2007). *Modeling, simulation and control of a hydraulic crane.* Växjö University .

Hiab Loglift 105. (n.d.). Retrieved Oct 06, 2011, from HIAB: http://www.hiab.com/Products/Forestry-and-recycling-cranes/Product-page/?parentProductGroupId=12790&productGroupId=12793&productId=18608

Huang J.Y and Gau C.Y. (2003). Modelling and designing a low-cost high fidelity mobile machine simulator. *International journal of humam computer studies*, 151-176.

Hung W.H, Kang S.C. (2009). Physcis-based crane model for the simulation of cooperative drections. *Proceedings of 9th international conference on construction application of virtual reality*, 237-246.

J R Juang,W H Hung,S C Kang. (2010). Using Game Physics for physics-based simulation. *Jounal of Information Technology in Construction*, 3-22.

J.Cragic, J. (2005). *Introduction to Robtics* (Third Edtion ed.). Pearson Education.

Jagadeesh Thati,Fazal Noorbasha. (2011). Crane Forwarder-control Algorithm for Automatic extension of prismatic link. *e-journal of Science & Technology*, 47-56.

M.Sigvardsson, T.Olsson. (2005). *Modelling and simulation of a Hydraulic crane.* Department of Technology,Högskolan Kalmar,Sweden.

Millington, I. (2007). *Game Physics Engine Development.* Morgan Kaufmann.

Paper and Wood insight. (n.d.). Retrieved from The paper and paperboard industry in the global market: http://www.forestindustries.fi/Pages/default.aspx

Python/C API Reference Manual. (n.d.). Retrieved October 03, 2011, from Python v2.7.2 documentation : http://docs.python.org/c-api/

Rigid Body Joint Constraint. (n.d.). Retrieved 11 6, 2011, from http://wiki.blender.org/index.php/Doc:2.5/Manual/Constraints/Relationship/Rigid_Body_Joint

Roosendall T,Wartmaan C. (2003). *The official blender gamekit:interactive 3D for artists.* San Francisco, USA: William Pollock.

Sheldon Andrews,Mohamad Eid2. (2007). Extending Blender:Development of Haptic Author Tool. *IEEE*.

Simith S.P,Duke D.J. (2000). user centered design and implementation of virtual environment. *International journal of human-computer studies,Vol.55,No.2*, 109-114.

Tony Mullen,Erwin Coumans. (2008). *Bounce, Tumble, and Splash!: Simulating the Physical World with Blender 3D.* John Wiley and Sons.

Transient response. (n.d.). Retrieved from Wikipedia: http://en.wikipedia.org/wiki/Transient_response

Trenholme D and Smith S.P. (2008). *Computer game engine for developing first-person virtual environments.* Virtual Reality.

Wei Guoqian; Zhang Zhenyan; Dong Haoming. (2009). Research and implement of the training simulator for overhead cranes. *Computer-Aided Industrial Design & Conceptual Design, 2009. CAID & CD 2009. IEEE 10th International Conference on* , (pp. 1049 - 1054). Wenzhou .

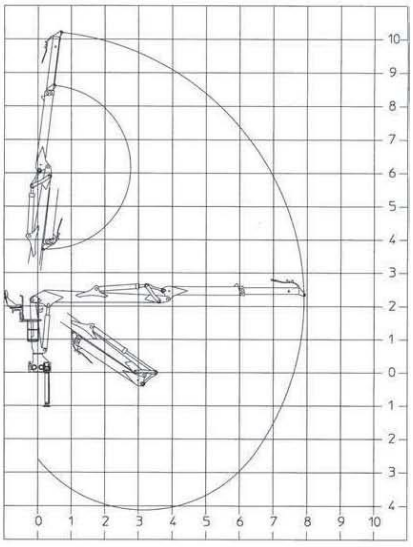
Whyte. (2002). *Virtual reality and built enviroment.* Oxford, UK: Architectural Press.

Appendix A: Data sheet

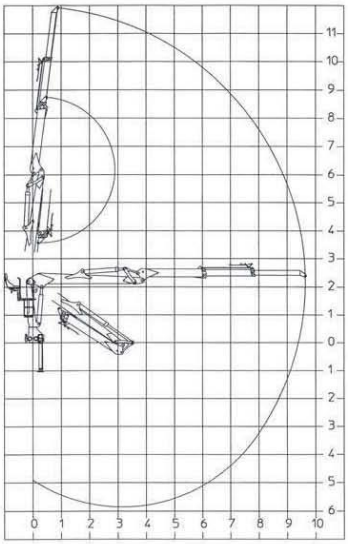
Hiab Loglift 105S Basic data

Technical data	S 79	ST 96
Max. lifting capacity (kNm)	106	101
Max outreach (m)	7.9	9.6
Hydraulic boom extensions (m)	1.6	3.2
Outreach / lifting capacity (m / kg)	3.0 / 3460	3.0 / 2250
	4.0 / 2600	4.0 / 2480
	5.0 / 2160	5.0 / 2040
	6.0 / 1800	6.0 / 1720
	7.0 / 1540	7.0 / 1470
	7.9 / 1370	8.0 / 1280
		9.0 / 1120
		9.6 / 1050
Recommended oil flow (l/min)	2x70-80	2x70-80
Power needed at rec. oil flow (kW)	34-68	34-68
Max. working pressure (MPa)	23	23
Slewing system	double	double
Slewing angle	425°	425°
Slewing torque, gross (kNm)	26.3	26.3
Weight without oil and grapple (kg)	1980	2110

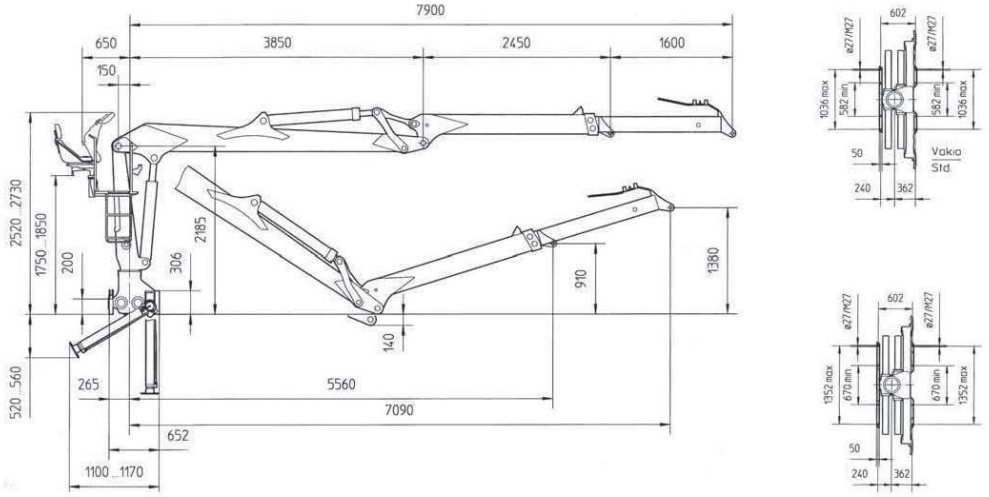
Hiab Loglift 105S 79



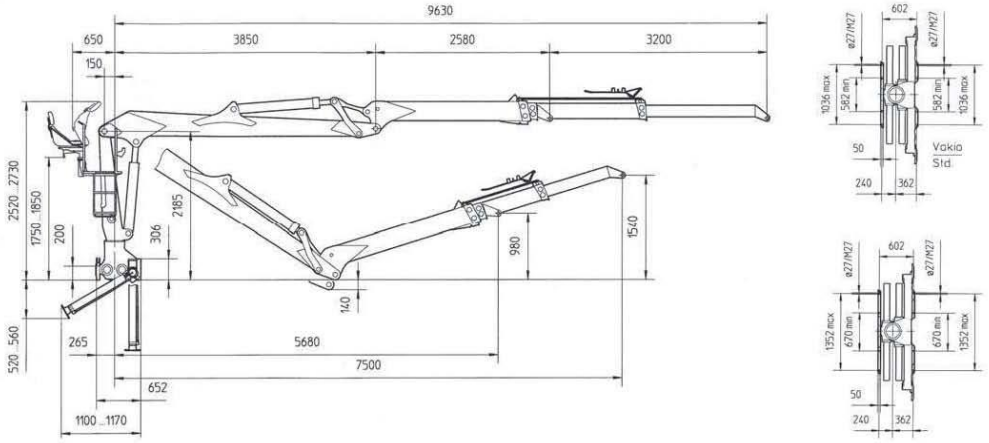
Hiab Loglift 105ST 96



Hiab Loglift 105S 79



Hiab Loglift 105ST 96



Appendix B: Matlab code of crane simulation

This appendix page will present the main Matlab-code. It could be divided into two parts, the main function defining the crane dynamics and the ode-function in which the extension of cylinder is defined. At the end of the *mainprogram.m*, the derivation of state space vector is calculated. Moreover, some leads corresponding some scripts to equations in thesis may help audience to have a better understanding of Matlab scripting based on Physics Model made in Chapter 2.

**********Start of Scripts**********

```
close all;
clear all;
syms theta1 theta2 theta1dd theta2dd;
syms theta1d theta2d;

%************************************
**********************************
%Dimensions of the crane
%************************************
**********************************
syms g l1 l2 l3 tl1 tl2 tl4 tl4_r tl5 tl6 j2_r rj1
rj2;
syms m1 m2;
syms Izz1 Izz2 Ixx1 Ixx2 Iyy1 Iyy2;
syms xc1 xc2 yc1 yc2;
syms c11x c11y c12x c12y c21x c21y c22x
c22y;

I_g = [0;g;0]; % abosolute value of gravity
C1I1 = [Ixx1 0 0
     0 Iyy1 0
     0 0 Izz1];
C2I2 = [Ixx2 0 0
     0 Iyy2 0
     0 0 Izz2];
P1c1 = [xc1;yc1;0;1];
P2c2 = [xc2;yc2;0;1];
q = [theta1;theta2];
qd = [theta1d;theta2d];
qdd = [theta1dd;theta2dd];
w1 = theta1d; % rotational velocity of link 2
w1_r = [0;0;w1];
w2 = theta2d+theta1d; % rotational velocity of
link 3
w2_r = [0;0;w2];
c1_1 = [c11x;c11y;0;1];
c1_2 = [c12x;c12y;0;1];
c2_1 = [c21x;c21y;0;1];
c2_2 = [c22x;c22y;0;1];
%************************************
**********************************
% Related the theta1(joint angle) to the stroke
xp1_real, also to the
% rotation angle from cylinder rod 1 to earth
frame.
% Then, we can get orientation of cylinder
force 1 related to earth frame.
%************************************
**********************************
syms ph0 r1 d1;
% ph0 = 0.7883 the initial value of ph1
% r1 = 140
% d1 = 575
ph1 = theta1 + ph0; %
xp1_real = sqrt(rj1^2+rj2^2-
2*cos(ph1)*rj1*rj2); % stroke of cylinder 1
beta1 = acos((xp1_real^2 + rj2^2 -
rj1^2)/(2*rj2*xp1_real));
A_c1 = - atan(r1/d1) - beta1 + pi/2 ;% rotation
from frame 0 to cylinder rod 1

%************************************
**********************************
% Related the theta2(joint angle) to the stroke
xp2_real, also to the
% rotation angle from cylinder rod 2 to earth
frame.
% Then, we can get orientation of cylinder
force 2 related to earth frame.
%************************************
**********************************
% r4 = 100;
```

```
% d4 = 400;
% r5 = 350;
% d5 = 242.27;
% r3 = 450-115.2;
% d3 = 1450;
% j2_r = [3.5;0.11520;0;1]; the joint connecting link 2 and link 3
% tl4_r = [0.65;0;0;1];  % torque link 4 in vector
syms r3 d3 r4 d4 r5 d5 xj2 yj2 xtl4 ytl4 tl1 tl2 tl4 tl5;
j2_r = [xj2;yj2;0;1];
tl4_r = [xtl4;ytl4;0;1];

% if theta2 is from 0 to 41.2911(0.7203 in rad) degree
% To find out of Angle of tl42 ( rotation angle from torque link 4 to major
% link 2) to orientate the force of cylinder 2 and position of cylinder 2.
% testing: theta2 = 41.2911*pi/180
% xp2 = sqrt(tl6^2+tl4^2-2*cos(alpxp2)*tl6*tl4)
alp3 = (pi/2 + theta2 + atan(r4/d4) + atan(d5/r5));
tl3 = sqrt(tl5^2 + tl1^2 -2*tl1*tl5*cos(alp3));
alp2 = acos((tl4^2 + tl3^2 - tl2^2)/(2*tl4*tl3));
beta2 = acos((tl5^2 + tl3^2 - tl1^2)/(2*tl5*tl3));
alpxp2 = pi - atan(r3/d3) - (- beta2 + alp2 - atan(r4/d4));  % angle corresponding to cylinder stroke 2
alpxp2 = simple(alpxp2);


% If theta2 is from 41.2911 to 180 degree
% To find out of Atl42,rotation angle from torque link 4 to major
% link 2, it is the orientation of cylinder 2 force related to earth frame.
alp3 = 2*pi - (pi/2 + theta2 + atan(r4/d4) + atan(d5/r5));
tl3 = sqrt(tl5^2 + tl1^2 -2*tl1*tl5*cos(alp3));
alp2 = acos((tl4^2 + tl3^2 - tl2^2)/(2*tl4*tl3));
beta2 = acos((tl5^2 + tl3^2 - tl1^2)/(2*tl5*tl3));
alpxp2 = pi - atan(r3/d3) - (beta2 + alp2 - atan(r4/d4)); % angle corresponding to cylinder stroke 2
alpxp2 = simple(alpxp2);

% The common part for different theta2 angle
syms alpxp2;
Ttl42 = dh(0,0,0,pi - atan(r3/d3)-alpxp2 ); % rotation from torque link 4 to major link 2
Atl42 = Ttl42(1:3,1:3);
Atl42 = simple(Atl42);
xp2_vec = j2_r + Ttl42*tl4_r - c2_1; % vector of cylinder 2 = end position of cylinder 2 - its start position
xp2_vec = simple(xp2_vec);
xp2_l = sqrt(xp2_vec(1)*xp2_vec(1) + xp2_vec(2)*xp2_vec(2)); % length of stroke of cylinder 2
A_c2 = atan(xp2_vec(2)/xp2_vec(1));
A_c2 = simple(A_c2); % rotation from cylinder 2 to earth frame

% testing (if theta2 > 48.6 degree)
% beta3 = acos((tl6^2 + xp2^2 - tl4^2)/(2*tl6*xp2));
% A_c2 = beta3 + atan(d3/r3) - pi/2;
% xp2 = sqrt(tl6^2+tl4^2-2*cos(alpxp2)*tl6*tl4);

%************************************************************************
% transformations from different items in the earth frame
%************************************************************************

%Transformation matrix and rotation matrix of joint
T01 = dh(0,0,0,theta1); %rotation from frame 0 to link 2
A01 = T01(1:3,1:3);
T12 = dh(0,l2,0,theta2 - pi);% rotation from frame 1 to link 3
A12 = T12(1:3,1:3);
T02 = T01*T12;% rotation from frame 0 to link 3
T02 = simple(T02);
A02 = T02(1:3,1:3);
A0c1 = A01*A_z(A_c1);
A0c1 = simple(A0c1);% rotation from frame 0 to end of cylinder rod 1
A0c2 = A01*A_z(A_c2); % rotation from frame 0 to end of cylinder 2

%************************************************************************
```

```
% Positions of cg of link2,3 and cylinders in
world frame
%************************************
*********************************
%Vectors in the world frame
p_l2_cg = T01*P1c1; % position of center of
gravity on link 2 in universal coordinator
p_l3_cg = T02*P2c2 ; % position of center of
gravity of link 3 in universal coordinator
p_l3_cg = simple(p_l3_cg);
p_c1_1 = T01*c1_1; % position of tube of
cylinder 1 in universal coordinator
p_c1_2 = T01*c1_2; % position of rod of
cylinder 1 in universal coordinator
p_c2_1 = T01*c2_1; % position of tube of
cylinder 2 in universal coordinator
p_c2_2 = T01*(j2_r + Ttl42*tl4_r); % rod point
of torque link 4 related to universal frame

%Jacobian of rod of cylinder 1 in the frame 0
J_c1_2 = jacobian(p_c1_2,q);
J_0_c1_2 = simple(J_c1_2);
%Jacobian of rod of cylinder 2 in the frame 0
J_c2_2 = jacobian(p_c2_2,q);
J_0_c2_2 = J_c2_2;

%************************************
*********************************
% Kinetic and Potential Energy
%************************************
*********************************
% velocities of center of gravity of link 2 and
link 3
v_l2_cg = jacobian(p_l2_cg,q)*qd;
v_l3_cg = jacobian(p_l3_cg,q)*qd;
v_l2_cg = v_l2_cg(1:3);
v_l3_cg = v_l3_cg(1:3);
% velocities^2
sqV01 =
v_l2_cg(1)^2+v_l2_cg(2)^2+v_l2_cg(3)^2;
sqV01 = simple(sqV01);
sqV02 =
v_l3_cg(1)^2+v_l3_cg(2)^2+v_l3_cg(3)^2;
sqV02 = simple(sqV02);

%Kinematic energy = rotaional energy + linear
energy
K1 = 1/2*m1*sqV01+1/2*w1_r.'*C1I1*w1_r;
K1 = simple(K1);
K2 = 1/2*m2*sqV02+1/2*w2_r.'*C2I2*w2_r;
K2 = simple(K2);
K = K1+K2;
K = simple(K);

% Potential energy
u1 = m1*I_g.'*p_l2_cg(1:3) ;
u2 = m2*I_g.'*p_l3_cg(1:3) ;
u = u1+u2;
u = simple(u);

% testing
% dk/dthetad
% Kthetad = simple(jacobian(K,qd));
% dKthetadq = jacobian((jacobian(K,qd)),q);
% dKthetadqd = jacobian((jacobian(K,qd)),qd);
% dk/dtheta
% Ktheta = simple(jacobian(K,q)).';
% du/dtheta
% utheta = simple(jacobian(u,q)).';
% total energy of crane
% Energy = dKthetadq*qd + dKthetadqd*qdd -
Ktheta + utheta = Q;
% Energy = simple(Energy);
% h = Q - (dKthetadq*qd - Ktheta + utheta);
% M = dKthetadqd;
% M = simple(M);
% qdd_r = inv(M)*h;

%************************************
*********************************
% Resulting cylinder force
%************************************
*********************************
syms f1 f2;
% Cylinder force at its local frame ( positive is
the x axis)
f1_r = [0;f1;0]; %at joint 1
f2_r = [f2;0;0]; %at joint 2

% Cylinder force in the frame 0(universal
frame)
I_0_f1 = A0c1*f1_r;
I_0_f1 = simple(I_0_f1);
I_0_f2 = A0c2*f2_r;

%************************************
*********************************
% Cylinder torque in earth frame
%************************************
*********************************
```

```
Q1 = J_0_c1_2(1:3,1:2).'*I_0_f1;
Q1 = simple(Q1);
% syms Q2x Q2y;
% Q2 = [Q2x;Q2y];
Q2 = J_0_c2_2(1:3,1:2).'*I_0_f2;
Q = Q1 + Q2;

%************************************
*********************************
% Get acceleration of theta1 and theta2-
theta1dd and theta2dd
%************************************
*********************************
% qdd_r = [theta1dd
%          theta2dd]
M = jacobian(jacobian(K,qd).',qd); %
M_function(theta2)
M = simple(M);
% h = Q - (dKthetadq*qd - Ktheta + utheta) for
%
h_function(theta1,theta1d,theta2,theta2d,f1,f2,al
pxp2)
h = - jacobian(jacobian(K,qd).',q) * qd +
jacobian(K,q).' - jacobian(u,q).' + Q ;
qdd_r = inv(M)*h;

%************************************
*********************************
% Call of ODE function
%************************************
*********************************
% plot the velocity and angle of crane
close all;
clear all;
hold on;
x0 = [0,0,0*pi/180,0]; % initial condition for
theta1,theta1d,theta2,theta2d
t_start = 0;
t_end = 9;
[t,x] = ode23(@forceode,[t_start,t_end],x0); %
ode45,try too
plot(t,x(:,1)*180/pi,'-',t,x(:,3)*180/pi,'-');
plot(y(:,1),y(:,2),'-',y(:,1),y(:,3),'-');
legend('Angular angle of Joint 1','Angular angle
of Joint 2');
xlabel('Time(s)');
ylabel('Rotation angle(degree)');
title('Angular displacement of two joints');


%---------end of ode---------------------------------
--------------------
% Appendex for dimensions of the crane

g = 9.81; % m/s^2
I_g = [0;g;0]; % abosolute value of gravity
l1 = 1.9 ;% link 1 = (1450+450)*10^-3
l2 = 3.9 ;% link 2 = 3900*10^-3;
l3 = 2.33; % link 3 = (4150-1820)*10^-3
tl1 = 0.4257; % torque link1 = sqrt((4142.37-
3900)^2+350^2)*10^-3;
tl2 = 0.5500; % torque link 2 =
sqrt((334.8+215.14)^2+(4150-
4142.27)^2)*10^-3;
tl4 = 0.65 ;% torque link 4 = sqrt((4142.27-
3500)^2+(215.14-115.2)^2)*10^-3;
tl4_r = [0.65;0;0;1];  % torque link 4 in vector
tl5 = 0.4123;% torque link 5 =
sqrt(400^2+100^2)*10^-3;
tl6 = 1.4882; % torque link 6 = sqrt((3500-
2050)^2+(450-115.2)^2)*10^-3; % tan(r3/d3) in
thesis
j2_r = [3.5;0.11520;0;1]; %the joint 2 in vector
rj1 = 0.3808; % links in cylinder 1 =
sqrt(350^2+150^2)*10^-3;
rj2 = 1.5598; % links in cylinder 1 =
sqrt(575^2+1450^2)*10^-3;
c1_1 = [0.575;-1.3;0;1]; % position of tube of
cylinder 1
c1_2 = [0.35;-0.15;0;1] ; % position of rod of
cylinder 1
c2_1 = [2.050;0.450;0;1]; % position of tube of
cylinder 2
c2_2 = [4.14227;0.21514;0;1]; % position of
rod of cylinder 2

m1 = 427.554 ; % mass of link 2
m2 = 238.981 ; % mass of link 3

c1_1 = [c11x;c11y;0;1]; % c12x = 0.35 c12y = -
0.15
c1_2 = [c12x;c12y;0;1];
c2_1 = [c21x;c21y;0;1];
c2_2 = [c22x;c22y;0;1];
```

**********End of Scripts**********

Appendix C: Python scripts for crane simulation

This Appendix page will display the Python scripts for developing joint angle 2 used to perform in Game Physics. It is divided into 4 major parts as explained in the previous section.

**********Start of Scripts**********

```
import bge
import mathutils

def main():

    # Declaration of all controller,scene or objects from
    # scenes.

   cont = bge.logic.getCurrentController()
   own = cont.owner
   rotation = own.actuators["rotation_rod2"]
   Up = own.sensors["Up_rod2"]
   scene = bge.logic.getCurrentScene()
   link3 = scene.objects["link3"]
   pin006 = scene.objects["joint_pin.006"]
   pin002 = scene.objects["joint_pin.002"]

   def init():

      # This part is used to initalise all the position of links
      # the initial angle of joint 2

      own['init_link3'] = mathutils.Vector((0.000786663,-3.033,1.729))

      own['init_pin002'] = mathutils.Vector((0,0.15,1.90))

      own['init_pin006'] = mathutils.Vector((0,-3.75,1.915))

      own['init_pin006tojoint1'] = - own['init_pin006'] + own['init_pin002']

      own['init_pin006tolink3'] = - own['init_pin006'] + own['init_link3']

      own['init_theta2']                                                    =
mathutils.Vector((own['init_pin006tolink3'][0],own['init_pin006tolink3'][1],own['init_pin006tolink3'][2])
).angle((own['init_pin006tojoint1'][0],own['init_pin006tojoint1'][1],own['init_pin006tojoint1'][2]))

   def update():
```

Page 84

```
    # display the position of rod 2 all the time
    # from this position, we calculate vector of link
      Cur_pin002 = pin002.worldPosition
      Cur_link3 =link3.worldPosition
      Cur_pin006 = pin006.worldPosition
      theta2_vector = Cur_link3 - Cur_pin006
      ref_theta2_vector = Cur_pin002 - Cur_pin006

      theta2 = mathutils.Vector((theta2_vector[0],theta2_vector[1],theta2_vector[2])).angle((ref_theta2_vector[0],ref_theta2_vector[1],ref_theta2_vector[2])) - own['init_theta2']

      own['theta2'] = theta2*180/3.141592

      Force = [0,0,-1875*3.14*4]
      own.applyForce(Force,True)
      cont.activate("rotation_rod2")
      print('theta2 is',own['theta2'])



  init()
  update()


main()
```

**********End of Scripts**********